TMS320DM643x DMP Ethernet Media Access Controller (EMAC)/ Management Data Input/Output (MDIO) Module

User's Guide

Literature Number: SPRU941A
April 2007

Texas Instruments

Contents

List of Figures

List of Tables

Read This First

About This Manual

This document provides a functional description of the Ethernet Media Access Controller (EMAC) and physical layer (PHY) device Management Data Input/Output (MDIO) module integrated in the TMS320DM643x Digital Media Processor (DMP). Included are the features of the EMAC and MDIO modules, a discussion of their architecture and operation, how these modules connect to the outside world, and the registers description for each module.

Notational Conventions

This document uses the following conventions.

- Hexadecimal numbers are shown with the suffix h. For example, the following number is 40 hexadecimal (decimal 64): 40h.
- Registers in this document are shown in figures and described in tables.
  - Each register figure shows a rectangle divided into fields that represent the fields of the register. Each field is labeled with its bit name, its beginning and ending bit numbers above, and its read/write properties below. A legend explains the notation used for the properties.
  - Reserved bits in a register figure designate a bit that is used for future device expansion.

Related Documentation From Texas Instruments

The following documents describe the TMS320DM643x Digital Media Processor (DMP). Copies of these documents are available on the Internet at www.ti.com. *Tip:* Enter the literature number in the search box provided at www.ti.com.

The current documentation that describes the DM643x DMP, related peripherals, and other technical collateral, is available in the C6000 DSP product folder at: www.ti.com/c6000.

SPRU978 — ***TMS320DM643x DMP DSP Subsystem Reference Guide.*** Describes the digital signal processor (DSP) subsystem in the TMS320DM643x Digital Media Processor (DMP).

SPRU983 — ***TMS320DM643x DMP Peripherals Overview Reference Guide.*** Provides an overview and briefly describes the peripherals available on the TMS320DM643x Digital Media Processor (DMP).

SPRAA84 — ***TMS320C64x to TMS320C64x+ CPU Migration Guide.*** Describes migrating from the Texas Instruments TMS320C64x digital signal processor (DSP) to the TMS320C64x+ DSP. The objective of this document is to indicate differences between the two cores. Functionality in the devices that is identical is not included.

SPRU732 — ***TMS320C64x/C64x+ DSP CPU and Instruction Set Reference Guide.*** Describes the CPU architecture, pipeline, instruction set, and interrupts for the TMS320C64x and TMS320C64x+ digital signal processors (DSPs) of the TMS320C6000 DSP family. The C64x/C64x+ DSP generation comprises fixed-point devices in the C6000 DSP platform. The C64x+ DSP is an enhancement of the C64x DSP with added functionality and an expanded instruction set.

SPRU871 — ***TMS320C64x+ DSP Megamodule Reference Guide.*** Describes the TMS320C64x+ digital signal processor (DSP) megamodule. Included is a discussion on the internal direct memory access (IDMA) controller, the interrupt controller, the power-down controller, memory protection, bandwidth management, and the memory and cache.

Ethernet Media Access Controller (EMAC)/ Management Data Input/Output (MDIO)

1 Introduction

This document provides a functional description of the Ethernet Media Access Controller (EMAC) and physical layer (PHY) device Management Data Input/Output (MDIO) module integrated in the TMS320DM643x Digital Media Processor (DMP). Included are the features of the EMAC and MDIO modules, a discussion of their architecture and operation, how these modules connect to the outside world, and a description of the registers for each module.

The EMAC controls the flow of packet data from the system to the PHY. The MDIO module controls PHY configuration and status monitoring.

Both the EMAC and the MDIO modules interface to the system core through a custom interface that allows efficient data transmission and reception. This custom interface is referred to as the EMAC control module and is considered integral to the EMAC/MDIO peripheral.

1.1 Purpose of the Peripheral

The EMAC module is used to move data between the DM643x device and another host connected to the same network, in compliance with the Ethernet protocol.

1.2 Features

The EMAC/MDIO has the following features:

- Synchronous 10/100 Mbps operation.
- Standard Media Independent Interface (MII) to physical layer device (PHY).
- EMAC acts as DMA master to either internal or external device memory space.
- Eight receive channels with VLAN tag discrimination for receive quality-of-service (QOS) support.
- Eight transmit channels with round-robin or fixed priority for transmit quality-of-service (QOS) support.
- Ether-Stats and 802.3-Stats statistics gathering.
- Transmit CRC generation selectable on a per channel basis.
- Broadcast frames selection for reception on a single channel.
- Multicast frames selection for reception on a single channel.
- Promiscuous receive mode frames selection for reception on a single channel (all frames, all good frames, short frames, error frames).
- Hardware flow control.
- 8K-byte local EMAC descriptor memory that allows the peripheral to operate on descriptors without affecting the CPU. The descriptor memory holds enough information to transfer up to 512 Ethernet packets without CPU intervention.
- Programmable interrupt logic permits the software driver to restrict the generation of back-to-back interrupts, which allows more work to be performed in a single call to the interrupt service routine.

1.3 Functional Block Diagram

Figure 1 shows the three main functional modules of the EMAC/MDIO peripheral:

- EMAC control module
- EMAC module
- MDIO module

The EMAC control module is the main interface between the device core processor and the EMAC module and MDIO module. The EMAC control module contains the necessary components to allow the EMAC to make efficient use of device memory, plus it controls device interrupts. The EMAC control module incorporates 8K-byte internal RAM to hold EMAC buffer descriptors.

The MDIO module implements the 802.3 serial management interface to interrogate and control up to 32 Ethernet PHYs connected to the device, using a shared two-wire bus. Host software uses the MDIO module to configure the autonegotiation parameters of each PHY attached to the EMAC, retrieve the negotiation results, and configure required parameters in the EMAC module for correct operation. The module is designed to allow almost transparent operation of the MDIO interface, with very little maintenance from the core processor.

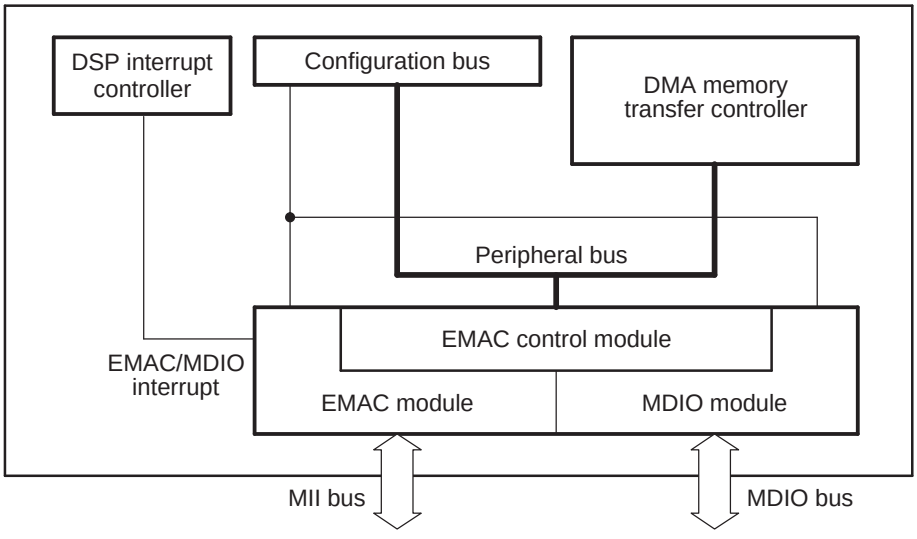
The EMAC module provides an efficient interface between the processor and the networked community. The EMAC on this device supports both 10Base-T (10 Mbits/sec), and 100BaseTX (100 Mbits/sec), in either half-duplex or full-duplex mode, with hardware flow control and quality-of-service (QOS) support.

Figure 1 also shows the main interface between the EMAC control module and the CPU. The following connections are made to the device core:

- The peripheral bus connection from the EMAC control module allows the EMAC module to read and write both internal and external memory through the DMA memory transfer controller.
- The EMAC control module, EMAC, and MDIO all have control registers. These registers are memory-mapped into device memory space via the device configuration bus. Along with these registers, the control module's internal RAM is mapped into this same range.
- The EMAC and MDIO interrupts are combined into a single interrupt within the control module. The interrupt from the control module then goes to the DSP interrupt controller.

The EMAC and MDIO interrupts are combined within the control module, so only the control module interrupt needs to be monitored by the application software or device driver. The combined EMAC/MDIO interrupt is connected to the DSP interrupt INT43, through the DSP interrupt controller.

Figure 1. EMAC and MDIO Block Diagram



1.4 Industry Standard(s) Compliance Statement

The EMAC peripheral conforms to the IEEE 802.3 standard, describing the Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer specifications. The IEEE 802.3 standard has also been adopted by ISO/IEC and re-designated as ISO/IEC 8802-3:2000(E).

In difference from this standard, the EMAC peripheral does not use the Transmit Coding Error signal MTXER. Instead of driving the error pin when an underflow condition occurs on a transmitted frame, the EMAC intentionally generates an incorrect checksum by inverting the frame CRC, so that the transmitted frame is detected as an error by the network.

2 Peripheral Architecture

This section discusses the architecture and basic function of the EMAC peripheral.

2.1 Clock Control

The frequencies for the transmit and receive clocks are fixed by the IEEE 802.3 specification as:

- 2.5 MHZ at 10 Mbps
- 25 MHZ at 100 Mbps

All EMAC logic is clocked synchronously with the PLL1/6 peripheral clock, except for the Ethernet MII synchronization logic.

The transmit and receive clock sources are provided from the external PHY via the MTCLK and MRCLK pins. These clocks are inputs to the EMAC module and operate at 2.5 MHZ in 10-Mbps mode, and at 25 MHZ in 100-Mbps mode. For timing purposes, data is transmitted and received with respect to MTCLK and MRCLK, respectively.

The MDIO clock is based on a divide-down of the peripheral clock (PLL1/6) specified to run up to 2.5 MHZ, although typical operation would be 1.0 MHZ. Since the peripheral clock frequency is variable (PLL1/6), the application software or driver controls the divide-down amount.

2.2 Memory Map

The EMAC peripheral includes internal memory that is used to hold information about the Ethernet packets received and transmitted. This internal RAM is 2K $\times$ 32 bits in size. Data can be written to and read from the EMAC internal memory by either the EMAC or the CPU. It is used to store buffer descriptors that are 4-words (16-bytes) deep. This 8K local memory holds enough information to transfer up to 512 Ethernet packets without CPU intervention.

The packet buffer descriptors can also be placed in the internal processor memory (L2), or in EMIF memory (DDR). There are some tradeoffs in terms of cache performance and throughput when descriptors are placed in the system memory, versus when they are placed in the EMAC's internal memory. Cache performance is improved when the buffer descriptors are placed in internal memory. However, the EMAC throughput is better when the descriptors are placed in the local EMAC RAM.

2.3 Signal Descriptions

Figure 2 shows a device with integrated EMAC and MDIO interfaced via a MII connection in a typical system. The EMAC module does not include a transmit error (MTXER) pin. In the case of transmit error, CRC inversion is used to negate the validity of the transmitted frame.

The individual EMAC and MDIO signals for the MII interface are summarized in Table 1. For more information, refer to either the IEEE 802.3 standard or ISO/IEC 8802-3:2000(E).

Figure 2. Typical Ethernet Configuration

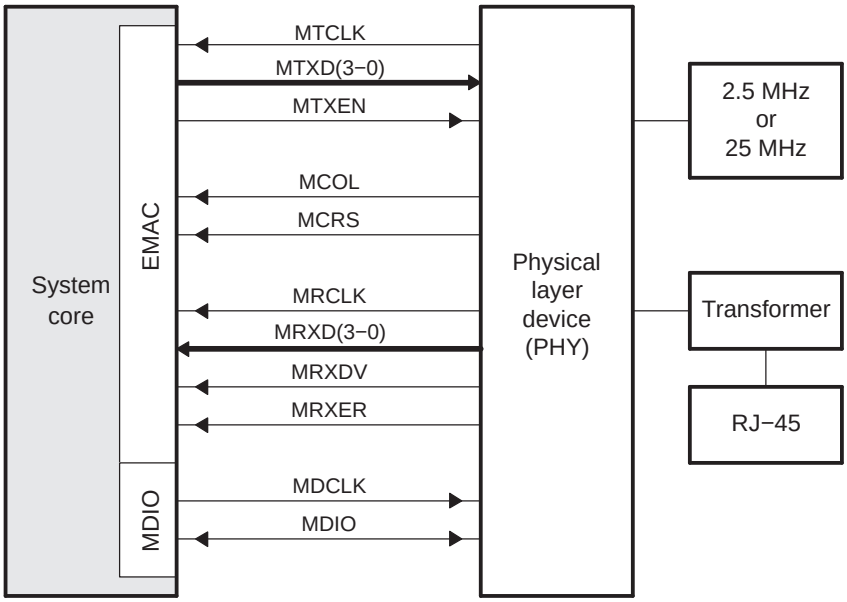


Table 1. EMAC and MDIO Signals

Signal	Type	Description
MTCLK	I	Transmit clock (MTCLK). The transmit clock is a continuous clock that provides the timing reference for transmit operations. The MTXD and MTXEN signals are tied to this clock. The clock is generated by the PHY and is 2.5 MHz at 10 Mbps operation and 25 MHz at 100 Mbps operation.
MTXD[3-0]	O	Transmit data (MTXD). The transmit data pins are a collection of 4 data signals comprising 4 bits of data. MTDX0 is the least-significant bit (LSB). The signals are synchronized by MTCLK and valid only when MTXEN is asserted.
MTXEN	O	Transmit enable (MTXEN). The transmit enable signal indicates that the MTXD pins are generating nibble data for use by the PHY. It is driven synchronously to MTCLK.
MCOL	I	Collision detected (MCOL). The MCOL pin is asserted by the PHY when it detects a collision on the network. It remains asserted while the collision condition persists. This signal is not necessarily synchronous to MTCLK nor MRCLK. This pin is used in half-duplex operation only.
MCRS	I	Carrier sense (MCRS). The MCRS pin is asserted by the PHY when the network is not idle in either transmit or receive. The pin is deasserted when both transmit and receive are idle. This signal is not necessarily synchronous to MTCLK nor MRCLK. This pin is used in half-duplex operation only.
MRCLK	I	Receive clock (MRCLK). The receive clock is a continuous clock that provides the timing reference for receive operations. The MRXD, MRXDV, and MRXER signals are tied to this clock. The clock is generated by the PHY and is 2.5 MHz at 10 Mbps operation and 25 MHz at 100 Mbps operation.
MRXD[3-0]	I	Receive data (MRXD). The receive data pins are a collection of 4 data signals comprising 4 bits of data. MRDX0 is the least-significant bit (LSB). The signals are synchronized by MRCLK and valid only when MRXDV is asserted.
MRXDV	I	Receive data valid (MRXDV). The receive data valid signal indicates that the MRXD pins are generating nibble data for use by the EMAC. It is driven synchronously to MRCLK.
MRXER	I	Receive error (MRXER). The receive error signal is asserted for one or more MRCLK periods to indicate that an error was detected in the received frame. This is meaningful only during data reception when MRXDV is active.
MDCLK	O	Management data clock (MDCLK). The MDIO data clock is sourced by the MDIO module on the system. It is used to synchronize MDIO data access operations done on the MDIO pin. The frequency of this clock is controlled by the CLKDIV bits in the MDIO control register (CONTROL).
MDIO	I/O	Management data input output (MDIO). The MDIO pin drives PHY management data into and out of the PHY by way of an access frame consisting of start of frame, read/write indication, PHY address, register address, and data bit cycles. The MDIO pin acts as an output for all but the data bit cycles at which time it is an input for read operations.

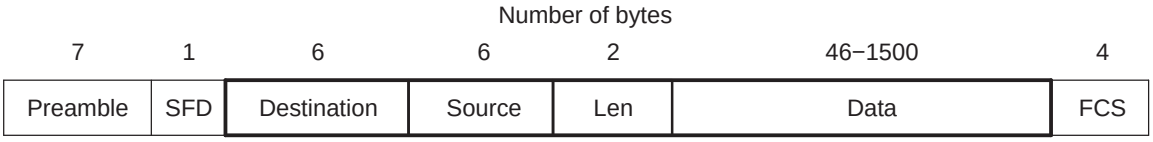
2.4 Ethernet Protocol Overview

A brief overview of the Ethernet protocol is given in the following subsections. For in-depth information on the Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method, which is the Ethernet's multiple access protocol, see the IEEE 802.3 standard document.

2.4.1 Ethernet Frame Format

All the Ethernet technologies use the same frame structure. The format of an Ethernet frame is shown in Figure 3 and described in Table 2. The Ethernet packet, which is the collection of bytes representing the data portion of a single Ethernet frame on the wire, is shown outlined in bold. The Ethernet frames are of variable lengths, with no frame smaller than 64 bytes or larger than RXMAXLEN bytes (header, data, and CRC).

Figure 3. Ethernet Frame Format



Legend: SFD=Start Frame Delimeter; FCS=Frame Check Sequence (CRC)

Table 2. Ethernet Frame Description

Field	Bytes	Description
Preamble	7	Preamble. These 7 bytes have a fixed value of 55h and serve to wake up the receiving EMAC ports and to synchronize their clocks to that of the sender's clock.
SFD	1	Start of Frame Delimiter. This field with a value of 5Dh immediately follows the preamble pattern and indicates the start of important data.
Destination	6	Destination address. This field contains the Ethernet MAC address of the EMAC port for which the frame is intended. It may be an individual or multicast (including broadcast) address. When the destination EMAC port receives an Ethernet frame with a destination address that does not match any of its MAC physical addresses, and no promiscuous, multicast or broadcast channel is enabled, it discards the frame.
Source	6	Source address. This field contains the MAC address of the Ethernet port that transmits the frame to the Local Area Network.
Len	2	Length/Type field. The length field indicates the number of EMAC client data bytes contained in the subsequent data field of the frame. This field can also be used to identify the type of data the frame is carrying.
Data	46 to (RXMAXLEN - 18)	Data field. This field carries the datagram containing the upper layer protocol frame, that is, IP layer datagram. The maximum transfer unit (MTU) of Ethernet is (RXMAXLEN - 18) bytes. This means that if the upper layer protocol datagram exceeds (RXMAXLEN - 18) bytes, then the host has to fragment the datagram and send it in multiple Ethernet packets. The minimum size of the data field is 46 bytes. This means that if the upper layer datagram is less then 46 bytes, the data field has to be extended to 46 bytes by appending extra bits after the data field, but prior to calculating and appending the FCS.
FCS	4	Frame Check Sequence. A cyclic redundancy check (CRC) is used by the transmit and receive algorithms to generate a CRC value for the FCS field. The frame check sequence covers the 60 to 1514 bytes of the packet data. Note that this 4-byte field may or may not be included as part of the packet data, depending on how the EMAC is configured.

2.4.2 Ethernet's Multiple Access Protocol

Nodes in an Ethernet Local Area Network are interconnected by a broadcast channel, as a result, when an EMAC port transmits a frame, all the adapters on the local network receive the frame. Carrier Sense Multiple Access with Collision Detection (CSMA/CD) algorithms are used when the EMAC operates in half-duplex mode. When operating in full-duplex mode, there is no contention for use of a shared medium, since there are exactly two ports on the local network.

Each port runs the CSMA/CD protocol without explicit coordination with the other ports on the Ethernet network. Within a specific port, the CSMA/CD protocol works as follows:

1. The port obtains data from upper layers protocols at its node, prepares an Ethernet frame, and puts the frame in a buffer.
2. If the port senses that the medium is idle it starts to transmit the frame. If the port senses that the transmission medium is busy, it waits until it senses no signal energy (plus an Inter-Packet Gap time) and then starts to transmit the frame.
3. While transmitting, the port monitors for the presence of signal energy coming from other ports. If the port transmits the entire frame without detecting signal energy from other Ethernet devices, the port is done with the frame.
4. If the port detects signal energy from other ports while transmitting, it stops transmitting its frame and instead transmits a 48-bit jam signal.
5. After transmitting the jam signal the port enters an exponential backoff phase. Specifically, when transmitting a given frame, after experiencing a number of collisions in a row for the frame, the port chooses a random value that is dependent on the number of collisions. The port then waits an amount of time that is multiple of this random value, and returns to step 2.

2.5 Programming Interface

2.5.1 Packet Buffer Descriptors

The buffer descriptor is a central part of the EMAC module and is how the application software describes Ethernet packets to be sent and empty buffers to be filled with incoming packet data. The basic descriptor format is shown in Figure 4 and described in Table 3.

For example, consider three packets to be transmitted, Packet A is a single fragment (60 bytes), Packet B is fragmented over three buffers (1514 bytes total), and Packet C is a single fragment (1514 bytes). The linked list of descriptors to describe these three packets is shown in Figure 5.

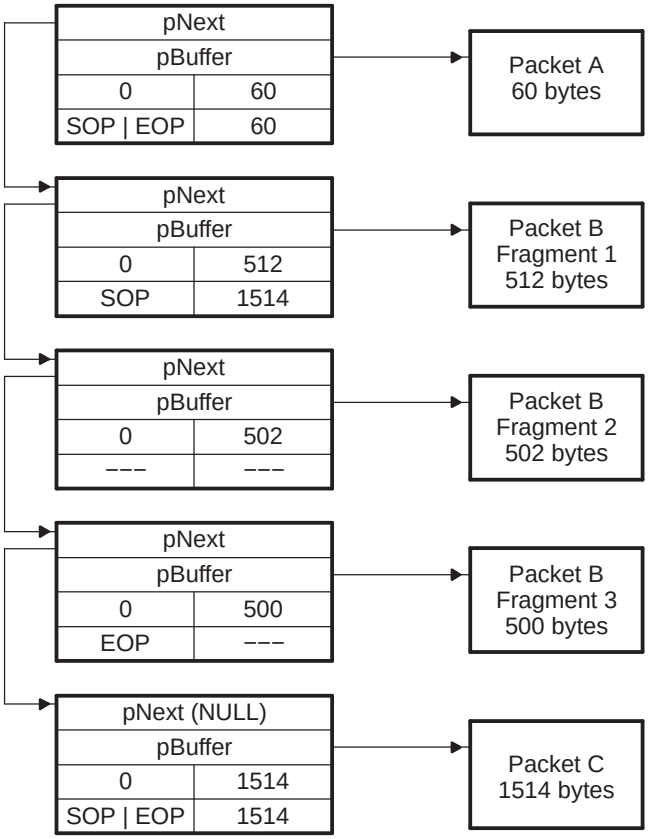
Figure 4. Basic Descriptor Format

	Bit Fields	
Word Offset	**31 16**	**15 0**
0	Next Descriptor Pointer	
1	Buffer Pointer	
2	Buffer Offset	Buffer Length
3	Flags	Packet Length

Table 3. Basic Descriptor Description

Word Offset	Field	Field Description
0	Next Descriptor Pointer	The next descriptor pointer is used to create a single linked list of descriptors. Each descriptor describes a packet or a packet fragment. When a descriptor points to a single buffer packet or the first fragment of a packet, the start of packet (SOP) flag is set in the flags field. When a descriptor points to a single buffer packet or the last fragment of a packet, the end of packet (EOP) flag is set. When a packet is fragmented, each fragment must have its own descriptor and appear sequentially in the descriptor linked list.
1	Buffer Pointer	The buffer pointer refers to the actual memory buffer that contains packet data during transmit operations, or is an empty buffer ready to receive packet data during receive operations.
2	Buffer Offset	The buffer offset is the offset from the start of the packet buffer to the first byte of valid data. This field only has meaning when the buffer descriptor points to a buffer that actually contains data.
	Buffer Length	The buffer length is the actual number of valid packet data bytes stored in the buffer. If the buffer is empty and waiting to receive data, this field represents the size of the empty buffer.
3	Flags	The flags field contains more information about the buffer, such as, is it the first fragment in a packet (SOP), the last fragment in a packet (EOP), or contains an entire contiguous Ethernet packet (both SOP and EOP). The flags are described in Section 2.5.4 and Section 2.5.5.
	Packet Length	The packet length only has meaning for buffers that both contain data and are the start of a new packet (SOP). In the case of SOP descriptors, the packet length field contains the length of the entire Ethernet packet, regardless if it is contained in a single buffer or fragmented over several buffers.

Figure 5. Typical Descriptor Linked List



2.5.2 Transmit and Receive Descriptor Queues

The EMAC module processes descriptors in linked list chains as discussed in Section 2.5.1. The lists controlled by the EMAC are maintained by the application software through the use of the head descriptor pointer registers (HDP). Since the EMAC supports eight channels for both transmit and receive, there are eight head descriptor pointer registers for both. They are:

- TX*n*HDP - Transmit Channel *n* DMA Head Descriptor Pointer Register
- RX*n*HDP - Receive Channel *n* DMA Head Descriptor Pointer Register

After an EMAC reset and before enabling the EMAC for send or receive, all 16 head descriptor pointer registers must be initialized to 0.

The EMAC uses a simple system to determine if a descriptor is currently owned by the EMAC or by the application software. There is a flag in the buffer descriptor flags called OWNER. When this flag is set, the packet that is referenced by the descriptor is considered to be owned by the EMAC. Note that ownership is done on a packet based granularity, not on descriptor granularity, so only SOP descriptors make use of the OWNER flag. As packets are processed, the EMAC patches the SOP descriptor of the corresponding packet and clears the OWNER flag. This is an indication that the EMAC has finished processing all descriptors up to and including the first with the EOP flag set, indicating the end of the packet (note this may only be one descriptor with both the SOP and EOP flags set).

To add a descriptor or a linked list of descriptors to an EMAC descriptor queue for the first time, the software application simply writes the pointer to the descriptor or first descriptor of a list to the corresponding HDP register. Note that the last descriptor in the list must have its “next” pointer cleared to 0. This is the only way the EMAC has of detecting the end of the list. So in the case where only a single descriptor is added, its “next descriptor” pointer must be initialized to 0.

The HDP must never be written to a second time while a previous list is active. To add additional descriptors to a descriptor list already owned by the EMAC, the NULL “next” pointer of the last descriptor of the previous list is patched with a pointer to the first descriptor in the new list. The list of new descriptors to be appended to the existing list must itself be NULL terminated before the pointer patch is performed.

There is a potential race condition where the EMAC may read the “next” pointer of a descriptor as NULL in the instant before an application appends additional descriptors to the list by patching the pointer. This case is handled by the software application always examining the buffer descriptor flags of all EOP packets, looking for a special flag called end of queue (EOQ). The EOQ flag is set by the EMAC on the last descriptor of a packet when the descriptor’s “next” pointer is NULL. This is the way the EMAC indicates to the software application that it believes it has reached the end of the list. When the software application sees the EOQ flag set, and there are more descriptors to process, the application may at that time submit the new list, or the portion of the appended list that was missed, by writing the new list pointer to the same HDP that started the process.

This process applies when adding packets to a transmit list, and empty buffers to a receive list.

2.5.3 Transmit and Receive EMAC Interrupts

The EMAC processes descriptors in linked list chains as discussed in Section 2.5.1, using the linked list queue mechanism discussed in Section 2.5.2.

The EMAC synchronizes descriptor list processing through the use of interrupts to the software application. The interrupts are controlled by the application using the interrupt masks, global interrupt enable, and the completion pointer register (CP). The CP is also called the interrupt acknowledge register.

As the EMAC supports eight channels for both transmit and receive, there are eight completion pointer registers for both. They are:

- TX*n*CP - Transmit Channel *n* Completion Pointer (Interrupt Acknowledge) Register
- RX*n*CP - Receive Channel *n* Completion Pointer (Interrupt Acknowledge) Register

These registers serve two purposes. When read, they return the pointer to the last descriptor that the EMAC has processed. When written by the software application, the value represents the last descriptor processed by the software application. When these two values do not match, the interrupt is active.

The system configuration determines whether or not an active interrupt actually interrupts the CPU. In general, the global interrupt for EMAC and MDIO must be enabled in the EMAC control module, and it also must be mapped in the DSP interrupt controller and enabled as a CPU interrupt. If the system is configured properly, the interrupt for a specific receive or transmit channel executes under the previously described conditions when the corresponding interrupt is enabled in the EMAC using the receive interrupt mask set register (RXINTMASKSET) or the transmit interrupt mask set register (TXINTMASKSET).

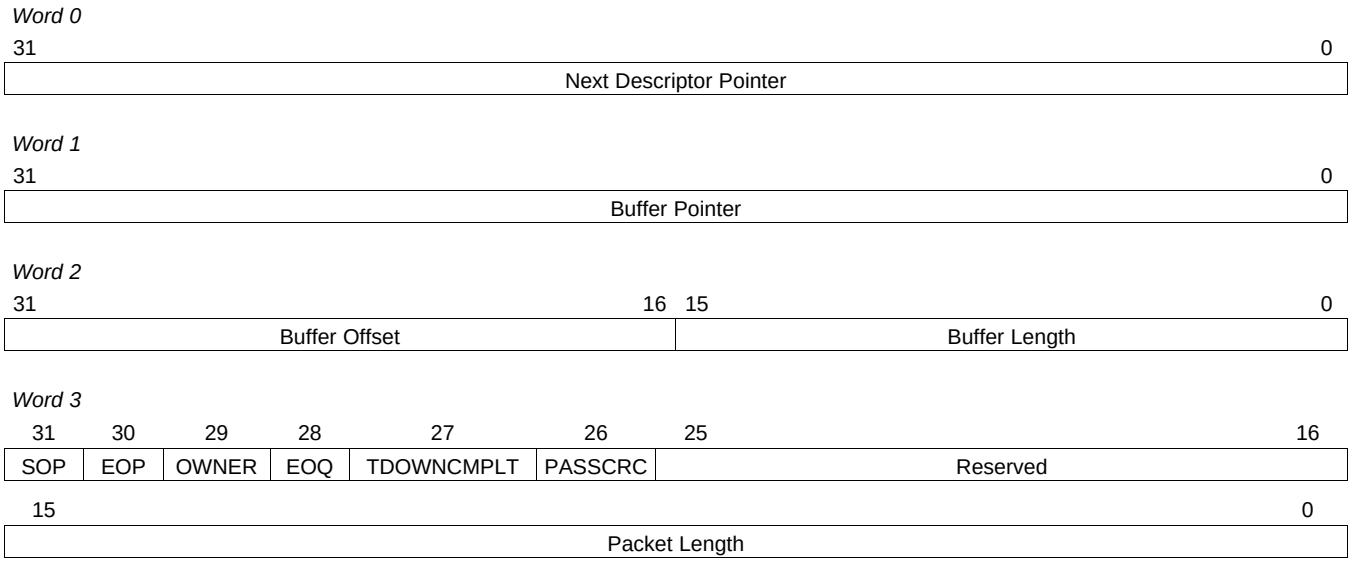
Whether or not the interrupt is enabled, the current state of the receive or transmit channel interrupt can be examined directly by the software application reading the receive interrupt status (unmasked) register (RXINTSTATRAW) and transmit interrupt status (unmasked) register (TXINTSTATRAW).

Interrupts are acknowledged when the application software updates the value of TX*n*CP or RX*n*CP with a value that matches the internal value kept by the EMAC. This mechanism ensures that the application software never misses an EMAC interrupt, since the interrupt and its acknowledgment are tied directly to the actual buffer descriptors processing.

2.5.4 Transmit Buffer Descriptor Format

A transmit (TX) buffer descriptor (Figure 6) is a contiguous block of four 32-bit data words aligned on a 32-bit boundary that describes a packet or a packet fragment. Example 1 shows the transmit buffer descriptor described by a C structure.

Figure 6. Transmit Buffer Descriptor Format



Example 1. Transmit Buffer Descriptor in C Structure Format

```
/*
// EMAC Descriptor
//
// The following is the format of a single buffer descriptor
// on the EM
AC.
*/
typedef struct _EMAC_Desc {
  struct _EMAC_Desc *pNext;     /* Pointer to next descriptor in chain  */
  Uint8             *pBuffer;   /* Pointer to data buffer               */
  Uint32            BufOffLen;  /* Buffer Offset(MSW) and Length(LSW)   */
  Uint32            PktFlgLen;  /* Packet Flags(MSW) and Length(LSW)    */
} EMAC_Desc;

/* Packet Flags                                                        */
#define EMAC_DSC_FLAG_SOP               0x80000000u
#define EMAC_DSC_FLAG_EOP               0x40000000u
#define EMAC_DSC_FLAG_OWNER             0x20000000u
#define EMAC_DSC_FLAG_EOQ               0x10000000u
#define EMAC_DSC_FLAG_TDOWNCMPLT        0x08000000u
#define EMAC_DSC_FLAG_PASSCRC           0x04000000u
```

2.5.4.1 Next Descriptor Pointer

The next descriptor pointer points to the 32-bit word aligned memory address of the next buffer descriptor in the transmit queue. This pointer is used to create a linked list of buffer descriptors. If the value of this pointer is zero, then the current buffer is the last buffer in the queue. The software application must set this value prior to adding the descriptor to the active transmit list. This pointer is not altered by the EMAC.

The value of pNext should never be altered once the descriptor is in an active transmit queue, unless its current value is NULL. If the pNext pointer is initially NULL, and more packets need to be queued for transmit, the software application may alter this pointer to point to a newly appended descriptor. The EMAC will use the new pointer value and proceed to the next descriptor unless the pNext value has already been read. In this latter case, the transmitter will halt on the transmit channel in question, and the software application may restart it at that time. The software can detect this case by checking for an end of queue (EOQ) condition flag on the updated packet descriptor when it is returned by the EMAC.

2.5.4.2 Buffer Pointer

The buffer pointer is the byte-aligned memory address of the memory buffer associated with the buffer descriptor. The software application must set this value prior to adding the descriptor to the active transmit list. This pointer is not altered by the EMAC.

2.5.4.3 Buffer Offset

This 16-bit field indicates how many unused bytes are at the start of the buffer. For example, a value of 0000h indicates that no unused bytes are at the start of the buffer and that valid data begins on the first byte of the buffer, while a value of 000Fh indicates that the first 15 bytes of the buffer are to be ignored by the EMAC and that valid buffer data starts on byte 16 of the buffer. The software application must set this value prior to adding the descriptor to the active transmit list. This field is not altered by the EMAC.

Note that this value is only checked on the first descriptor of a given packet (where the start of packet (SOP) flag is set). It can not be used to specify the offset of subsequent packet fragments. Also, since the buffer pointer may point to any byte–aligned address, this field may be entirely superfluous, depending on the device driver architecture.

The range of legal values for this field is 0 to (Buffer Length – 1).

2.5.4.4 Buffer Length

This 16-bit field indicates how many valid data bytes are in the buffer. On single fragment packets, this value is also the total length of the packet data to be transmitted. If the buffer offset field is used, the offset bytes are not counted as part of this length. This length counts only valid data bytes. The software application must set this value prior to adding the descriptor to the active transmit list. This field is not altered by the EMAC.

2.5.4.5 Packet Length

This 16-bit field specifies the number of data bytes in the entire packet. Any leading buffer offset bytes are not included. The sum of the buffer length fields of each of the packet's fragments (if more than one) must be equal to the packet length. The software application must set this value prior to adding the descriptor to the active transmit list. This field is not altered by the EMAC. This value is only checked on the first descriptor of a given packet (where the start of packet (SOP) flag is set).

2.5.4.6 Start of Packet (SOP) Flag

When set, this flag indicates that the descriptor points to a packet buffer that is the start of a new packet. In the case of a single fragment packet, both the SOP and end of packet (EOP) flags are set. Otherwise, the descriptor pointing to the last packet buffer for the packet sets the EOP flag. This bit is set by the software application and is not altered by the EMAC.

2.5.4.7 End of Packet (EOP) Flag

When set, this flag indicates that the descriptor points to a packet buffer that is last for a given packet. In the case of a single fragment packet, both the start of packet (SOP) and EOP flags are set. Otherwise, the descriptor pointing to the last packet buffer for the packet sets the EOP flag. This bit is set by the software application and is not altered by the EMAC.

2.5.4.8 Ownership (OWNER) Flag

When set, this flag indicates that all the descriptors for the given packet (from SOP to EOP) are currently owned by the EMAC. This flag is set by the software application on the SOP packet descriptor before adding the descriptor to the transmit descriptor queue. For a single fragment packet, the SOP, EOP, and OWNER flags are all set. The OWNER flag is cleared by the EMAC once it is finished with all the descriptors for the given packet. Note that this flag is valid on SOP descriptors only.

2.5.4.9 End of Queue (EOQ) Flag

When set, this flag indicates that the descriptor in question was the last descriptor in the transmit queue for a given transmit channel, and that the transmitter has halted. This flag is initially cleared by the software application prior to adding the descriptor to the transmit queue. This bit is set by the EMAC when the EMAC identifies that a descriptor is the last for a given packet (the EOP flag is set), and there are no more descriptors in the transmit list (next descriptor pointer is NULL).

The software application can use this bit to detect when the EMAC transmitter for the corresponding channel has halted. This is useful when the application appends additional packet descriptors to a transmit queue list that is already owned by the EMAC. Note that this flag is valid on EOP descriptors only.

2.5.4.10 Teardown Complete (TDOWNCMPLT) Flag

This flag is used when a transmit queue is being torn down, or aborted, instead of allowing it to be transmitted. This would happen under device driver reset or shutdown conditions. The EMAC sets this bit in the SOP descriptor of each packet as it is aborted from transmission.

Note that this flag is valid on SOP descriptors only. Also note that only the first packet in an unsent list has the TDOWNCMPLT flag set. Subsequent descriptors are not even processed by the EMAC.

2.5.4.11 Pass CRC (PASSCRC) Flag

This flag is set by the software application in the SOP packet descriptor before it adds the descriptor to the transmit queue. Setting this bit indicates to the EMAC that the 4 byte Ethernet CRC is already present in the packet data, and that the EMAC should not generate its own version of the CRC.

When the CRC flag is cleared, the EMAC generates and appends the 4-byte CRC. The buffer length and packet length fields do not include the CRC bytes. When the CRC flag is set, the 4-byte CRC is supplied by the software application and is already appended to the end of the packet data. The buffer length and packet length fields include the CRC bytes, as they are part of the valid packet data. Note that this flag is valid on SOP descriptors only.

2.5.5 Receive Buffer Descriptor Format

A receive (RX) buffer descriptor (Figure 7) is a contiguous block of four 32-bit data words aligned on a 32-bit boundary that describes a packet or a packet fragment. Example 2 shows the receive buffer descriptor described by a C structure.

2.5.5.1 Next Descriptor Pointer

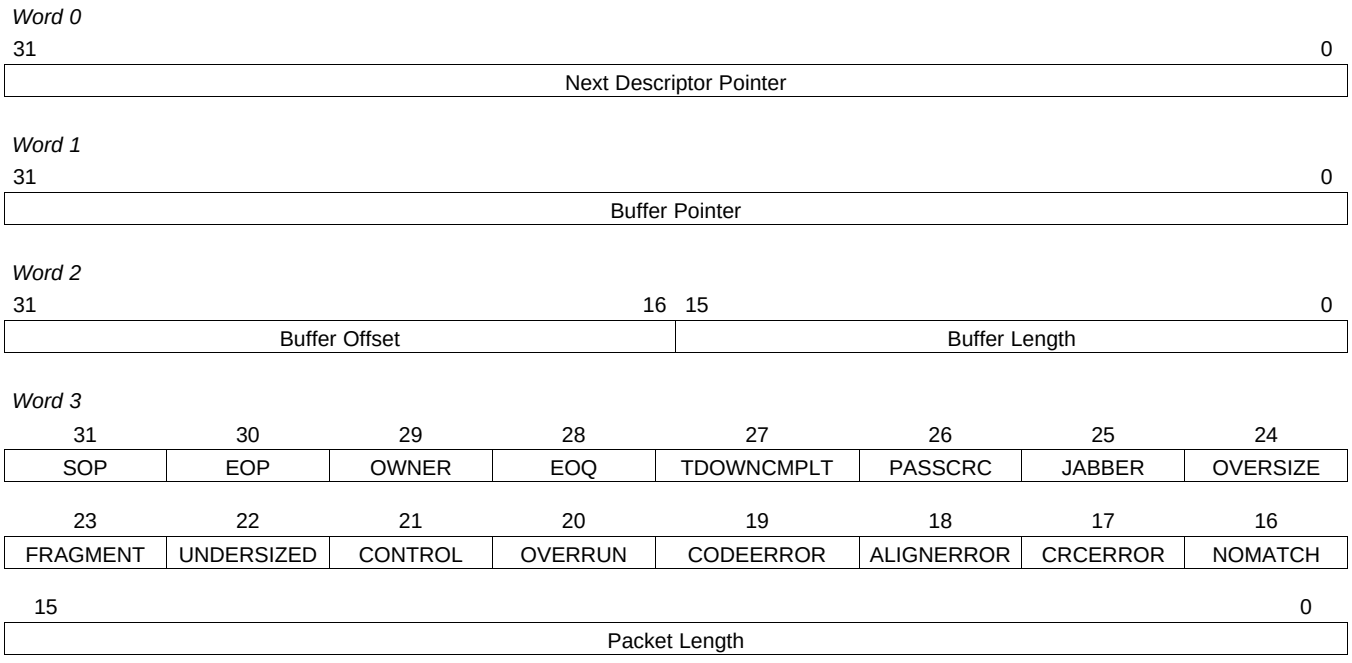
This pointer points to the 32–bit word aligned memory address of the next buffer descriptor in the receive queue. This pointer is used to create a linked list of buffer descriptors. If the value of this pointer is zero, then the current buffer is the last buffer in the queue. The software application must set this value prior to adding the descriptor to the active receive list. This pointer is not altered by the EMAC.

The value of pNext should never be altered once the descriptor is in an active receive queue, unless its current value is NULL. If the pNext pointer is initially NULL, and more empty buffers can be added to the pool, the software application may alter this pointer to point to a newly appended descriptor. The EMAC will use the new pointer value and proceed to the next descriptor unless the pNext value has already been read. In this latter case, the receiver will halt the receive channel in question, and the software application may restart it at that time. The software can detect this case by checking for an end of queue (EOQ) condition flag on the updated packet descriptor when it is returned by the EMAC.

2.5.5.2 Buffer Pointer

The buffer pointer is the byte-aligned memory address of the memory buffer associated with the buffer descriptor. The software application must set this value prior to adding the descriptor to the active receive list. This pointer is not altered by the EMAC.

Figure 7. Receive Buffer Descriptor Format



Example 2. Receive Buffer Descriptor in C Structure Format

```
/*
// EMAC Descriptor
//
// The following is the format of a single buffer descriptor
// on the EMAC.
*/
typedef struct _EMAC_Desc {
  struct _EMAC_Desc *pNext;     /* Pointer to next descriptor in chain  */
  Uint8             *pBuffer;   /* Pointer to data buffer               */
  Uint32            BufOffLen;  /* Buffer Offset(MSW) and Length(LSW)   */
  Uint32
PktFlgLen;  /* Packet Flags(MSW) and Length(LSW)    */
} EMAC_Desc;

/* Packet Flags                                                         */
#define EMAC_DSC_FLAG_SOP               0x80000000u
#define EMAC_DSC_FLAG_EOP               0x40000000u
#define EMAC_DSC_FLAG_OWNER             0x20000000u
#define EMAC_DSC_FLAG_EOQ               0x10000000u
#define EMAC_DSC_FLAG_TDOWNCMPLT        0x08000000u
#define EMAC_DSC_FLAG_PASSCRC           0x04000000u
#define EMAC_DSC_FLAG_JABBER            0x02000000u
#define EMAC_DSC_FLAG_OVERSIZE          0x01000000u
#define EMAC_DSC_FLAG_FRAGMENT          0x00800000u
#define EMAC_DSC_FLAG_UNDERSIZED        0x00400000u
#define EMAC_DSC_FLAG_CONTROL           0x00200000u
#define EMAC_DSC_FLAG_OVERRUN           0x00100000u
#define EMAC_DSC_FLAG_CODEERROR         0x00080000u
#define EMAC_DSC_FLAG_ALIGNERROR        0x00040000u
#define EMAC_DSC_FLAG_CRCERROR          0x00020000u
#define EMAC_DSC_FLAG_NOMATCH           0x00010000u
```

2.5.5.3 Buffer Offset

This 16-bit field must be initialized to zero by the software application before adding the descriptor to a receive queue.

Whether or not this field is updated depends on the setting of the RXBUFFEROFFSET register. When the offset register is set to a non-zero value, the received packet is written to the packet buffer at an offset given by the value of the register, and this value is also written to the buffer offset field of the descriptor.

When a packet is fragmented over multiple buffers because it does not fit in the first buffer supplied, the buffer offset only applies to the first buffer in the list, which is where the start of packet (SOP) flag is set in the corresponding buffer descriptor. In other words, the buffer offset field is only updated by the EMAC on SOP descriptors.

The range of legal values for the BUFFEROFFSET register is 0 to (Buffer Length – 1) for the smallest value of buffer length for all descriptors in the list.

2.5.5.4 Buffer Length

This 16-bit field is used for two purposes:

- Before the descriptor is first placed on the receive queue by the application software, the buffer length field is first initialized by the software to have the physical size of the empty data buffer pointed to by the buffer pointer field.
- After the empty buffer has been processed by the EMAC and filled with received data bytes, the buffer length field is updated by the EMAC to reflect the actual number of valid data bytes written to the buffer.

2.5.5.5 Packet Length

This 16-bit field specifies the number of data bytes in the entire packet. This value is initialized to zero by the software application for empty packet buffers. The value is filled in by the EMAC on the first buffer used for a given packet. This is signified by the EMAC setting a start of packet (SOP) flag. The packet length is set by the EMAC on all SOP buffer descriptors.

2.5.5.6 Start of Packet (SOP) Flag

When set, this flag indicates that the descriptor points to a packet buffer that is the start of a new packet. In the case of a single fragment packet, both the SOP and end of packet (EOP) flags are set. Otherwise, the descriptor pointing to the last packet buffer for the packet has the EOP flag set. This flag is initially cleared by the software application before adding the descriptor to the receive queue. This bit is set by the EMAC on SOP descriptors.

2.5.5.7 End of Packet (EOP) Flag

When set, this flag indicates that the descriptor points to a packet buffer that is last for a given packet. In the case of a single fragment packet, both the start of packet (SOP) and EOP flags are set. Otherwise, the descriptor pointing to the last packet buffer for the packet has the EOP flag set. This flag is initially cleared by the software application before adding the descriptor to the receive queue. This bit is set by the EMAC on EOP descriptors.

2.5.5.8 Ownership (OWNER) Flag

When set, this flag indicates that the descriptor is currently owned by the EMAC. This flag is set by the software application before adding the descriptor to the receive descriptor queue. This flag is cleared by the EMAC once it is finished with a given set of descriptors, associated with a received packet. The flag is updated by the EMAC on SOP descriptor only. So when the application identifies that the OWNER flag is cleared on an SOP descriptor, it may assume that all descriptors up to and including the first with the EOP flag set have been released by the EMAC. (Note that in the case of single buffer packets, the same descriptor will have both the SOP and EOP flags set.)

2.5.5.9 End of Queue (EOQ) Flag

When set, this flag indicates that the descriptor in question was the last descriptor in the receive queue for a given receive channel, and that the corresponding receiver channel has halted. This flag is initially cleared by the software application prior to adding the descriptor to the receive queue. This bit is set by the EMAC when the EMAC identifies that a descriptor is the last for a given packet received (also sets the EOP flag), and there are no more descriptors in the receive list (next descriptor pointer is NULL).

The software application can use this bit to detect when the EMAC receiver for the corresponding channel has halted. This is useful when the application appends additional free buffer descriptors to an active receive queue. Note that this flag is valid on EOP descriptors only.

2.5.5.10 Teardown Complete (TDOWNCMPLT) Flag

This flag is used when a receive queue is being torn down, or aborted, instead of being filled with received data. This would happen under device driver reset or shutdown conditions. The EMAC sets this bit in the descriptor of the first free buffer when the tear down occurs. No additional queue processing is performed.

2.5.5.11 Pass CRC (PASSCRC) Flag

This flag is set by the EMAC in the SOP buffer descriptor if the received packet includes the 4-byte CRC. This flag should be cleared by the software application before submitting the descriptor to the receive queue.

2.5.5.12 Jabber Flag

This flag is set by the EMAC in the SOP buffer descriptor, if the received packet is a jabber frame and was not discarded because the RXCEFEN bit was set in the RXMBPENABLE. Jabber frames are frames that exceed the RXMAXLEN in length, and have CRC, code, or alignment errors.

2.5.5.13 Oversize Flag

This flag is set by the EMAC in the SOP buffer descriptor, if the received packet is an oversized frame and was not discarded because the RXCEFEN bit was set in the RXMBPENABLE.

2.5.5.14 Fragment Flag

This flag is set by the EMAC in the SOP buffer descriptor, if the received packet is only a packet fragment and was not discarded because the RXCEFEN bit was set in the RXMBPENABLE.

2.5.5.15 Undersized Flag

This flag is set by the EMAC in the SOP buffer descriptor, if the received packet is undersized and was not discarded because the RXCSFEN bit was set in the RXMBPENABLE.

2.5.5.16 Control Flag

This flag is set by the EMAC in the SOP buffer descriptor, if the received packet is an EMAC control frame and was not discarded because the RXCMFEN bit was set in the RXMBPENABLE.

2.5.5.17 Overrun Flag

This flag is set by the EMAC in the SOP buffer descriptor, if the received packet was aborted due to a receive overrun.

2.5.5.18 Code Error (CODEERROR) Flag

This flag is set by the EMAC in the SOP buffer descriptor, if the received packet contained a code error and was not discarded because the RXCEFEN bit was set in the RXMBPENABLE.

2.5.5.19 Alignment Error (ALIGNERROR) Flag

This flag is set by the EMAC in the SOP buffer descriptor, if the received packet contained an alignment error and was not discarded because the RXCEFEN bit was set in the RXMBPENABLE.

2.5.5.20 CRC Error (CRCERROR) Flag

This flag is set by the EMAC in the SOP buffer descriptor, if the received packet contained a CRC error and was not discarded because the RXCEFEN bit was set in the RXMBPENABLE.

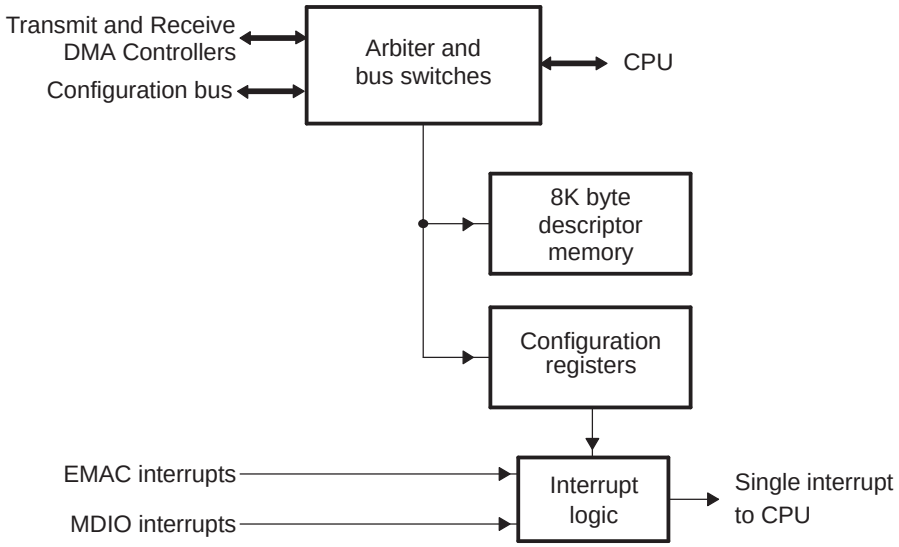
2.5.5.21 No Match (NOMATCH) Flag

This flag is set by the EMAC in the SOP buffer descriptor, if the received packet did not pass any of the EMAC's address match criteria and was not discarded because the RXCAFEN bit was set in the RXMBPENABLE. Although the packet is a valid Ethernet data packet, it was only received because the EMAC is in promiscuous mode.

2.6 EMAC Control Module

The basic functions of the EMAC control module (Figure 8) are to interface the EMAC and MDIO modules to the rest of the system, and to provide for a local memory space to hold EMAC packet buffer descriptors. Local memory is used to help avoid contention to device memory spaces. Other functions include the bus arbiter, and interrupt logic control.

Figure 8. EMAC Control Module Block Diagram



2.6.1 Internal Memory

The EMAC control module includes 8K bytes of internal memory. The internal memory block is essential for allowing the EMAC to operate more independently of the CPU. It also prevents memory underflow conditions when the EMAC issues read or write requests to descriptor memory. (Memory accesses to read or write the actual Ethernet packet data are protected by the EMAC's internal FIFOs).

A descriptor is a 16-byte memory structure that holds information about a single Ethernet packet buffer, which may contain a full or partial Ethernet packet. Thus with the 8K memory block provided for descriptor storage, the EMAC module can send and received up to a combined 512 packets before it needs to be serviced by application or driver software.

2.6.2 Bus Arbiter

The EMAC control module bus arbiter operates transparently to the rest of the system. It is used:

- To arbitrate between the CPU and EMAC buses for access to internal descriptor memory.
- To arbitrate between internal EMAC buses for access to system memory.

2.6.3 Interrupt Control

The EMAC control module combines multiple interrupt conditions generated by the EMAC and MDIO modules into a single interrupt signal that is mapped to a CPU interrupt via the CPU interrupt controller. The control module uses two registers to control the interrupt signal to the CPU:

- The INTEN bit in the EMAC control module interrupt control register (EWCTL) globally enables and disables the interrupt signal to the CPU. The INTEN bit is used to drive the interrupt line low during interrupt processing so that upon reenabling the bit, the interrupt signal will rise if another interrupt condition exists; thus, creating a rising edge detectable by the CPU.
- The EMAC control module interrupt timer count register (EWINTTCNT) is programmed with a value (EWINTTCNT) that counts down once EMAC/MDIO interrupts are enabled using EWCTL. The CPU interrupt signal is prevented from rising again until this count reaches 0.

EWINTTCNT has no effect on interrupts once the count reaches 0, so there is no induced interrupt latency on random sporadic interrupts. However, the count delays the issuance of a second interrupt immediately after a first. This protects the system from getting into an interrupt thrashing mode where the software interrupt service routine (ISR) completes processing just in time to get another interrupt. By postponing subsequent interrupts in a back-to-back condition, the software application or driver can get more work done in its ISR.

EWINTTCNT reset value can be adjusted from within the ISR according to current system load, or simply set to a fixed value that assures a maximum number of interrupts per second.

The counter counts at the peripheral clock frequency of PLL1/6; its default reset count is 0 (inactive), its maximum value is 1 FFFFh (131 071).

2.7 MDIO Module

The MDIO module is used to manage up to 32 physical layer (PHY) devices connected to the Ethernet Media Access Controller (EMAC). The DM643x device supports a single PHY being connected to the EMAC at any given time. The MDIO module is designed to allow almost transparent operation of the MDIO interface with little maintenance from the CPU.

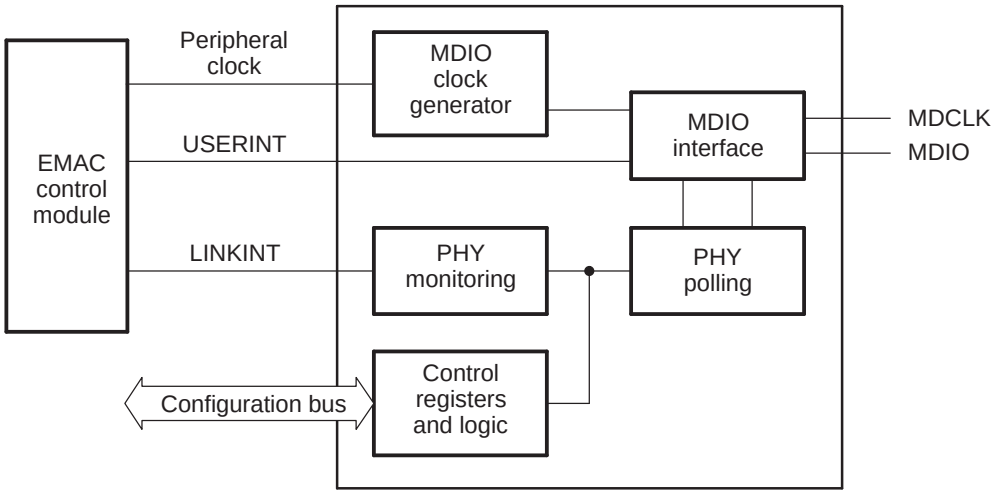
The MDIO module continuously polls 32 MDIO addresses in order to enumerate all PHY devices in the system. Once a PHY device has been detected, the MDIO module reads the MDIO PHY link status register (LINK) to monitor the PHY link state. Link change events are stored in the MDIO module, which can interrupt the CPU. This storing of the events allows the CPU to poll the link status of the PHY device without continuously performing MDIO module accesses. However, when the CPU must access the MDIO module for configuration and negotiation, the MDIO module performs the MDIO read or write operation independent of the CPU. This independent operation allows the processor to poll for completion or interrupt the CPU once the operation has completed.

2.7.1 MDIO Module Components

The MDIO module (Figure 9) interfaces to the PHY components through two MDIO pins (MDCLK and MDIO), and to the CPU through the EMAC control module and the configuration bus. The MDIO module consists of the following logical components:

- MDIO clock generator
- Global PHY detection and link state monitoring
- Active PHY monitoring
- PHY register user access

Figure 9. MDIO Module Block Diagram



2.7.1.1 MDIO Clock Generator

The MDIO clock generator controls the MDIO clock based on a divide-down of the peripheral clock (PLL1/6) in the EMAC control module. The MDIO clock is specified to run up to 2.5 MHZ, although typical operation would be 1.0 MHZ. Since the peripheral clock frequency is variable (PLL1/6), the application software or driver controls the divide-down amount.

2.7.1.2 Global PHY Detection and Link State Monitoring

The MDIO module continuously polls all 32 MDIO addresses in order to enumerate the PHY devices in the system. The module tracks whether or not a PHY on a particular address has responded, and whether or not the PHY currently has a link. Using this information allows the software application to quickly determine which MDIO address the PHY is using.

2.7.1.3 Active PHY Monitoring

Once a PHY candidate has been selected for use, the MDIO module transparently monitors its link state by reading the MDIO PHY link status register (LINK). Link change events are stored on the MDIO device and can optionally interrupt the CPU. This allows the system to poll the link status of the PHY device without continuously performing costly MDIO accesses.

2.7.1.4 PHY Register User Access

When the CPU must access MDIO for configuration and negotiation, the PHY access module performs the actual MDIO read or write operation independent of the CPU. This allows the CPU to poll for completion or receive an interrupt when the read or write operation has been performed. The user access registers USERACCESS*n* allows the software to submit the access requests for the PHY connected to the device.

2.7.2 MDIO Module Operational Overview

The MDIO module implements the 802.3 serial management interface to interrogate and control an Ethernet PHY, using a shared two-wired bus. It separately performs autodetection and records the current link status of up to 32 PHYs, polling all 32 MDIO addresses.

Application software uses the MDIO module to configure the autonegotiation parameters of the PHY attached to the EMAC, retrieve the negotiation results, and configure required parameters in the EMAC.

In this device, the Ethernet PHY attached to the system can be directly controlled and queried. The Media Independent Interface (MII) address of this PHY device is specified in one of the PHYADRMON bits in the MDIO user PHY select register (USERPHYSEL*n*). The MDIO module can be programmed to trigger a CPU interrupt on a PHY link change event, by setting the LINKINTENB bit in USERPHYSEL*n*. Reads and writes to registers in this PHY device are performed using the MDIO user access register (USERACCESS*n*).

The MDIO module powers-up in an idle state until specifically enabled by setting the ENABLE bit in the MDIO control register (CONTROL). At this time, the MDIO clock divider and preamble mode selection are also configured. The MDIO preamble is enabled by default, but can be disabled when the connected PHY does not require it. Once the MDIO module is enabled, the MDIO interface state machine continuously polls the PHY link status (by reading the generic status register) of all possible 32 PHY addresses and records the results in the MDIO PHY alive status register (ALIVE) and MDIO PHY link status register (LINK). The corresponding bit for the connected PHY (0-31) is set in ALIVE, if the PHY responded to the read request. The corresponding bit is set in LINK, if the PHY responded and also is currently linked. In addition, any PHY register read transactions initiated by the application software using USERACCESS*n* causes ALIVE to be updated.

The USERPHYSEL*n* is used to track the link status of the connected PHY address. A change in the link status of the PHY being monitored sets the appropriate bit in the MDIO link status change interrupt registers (LINKINTRAW and LINKINTMASKED), if enabled by the LINKINTENB bit in USERPHYSEL*n*.

While the MDIO module is enabled, the host issues a read or write transaction over the MII management interface using the DATA, PHYADR, REGADR, and WRITE bits in USERACCESS*n*. When the application sets the GO bit in USERACCESS*n*, the MDIO module begins the transaction without any further intervention from the CPU. Upon completion, the MDIO module clears the GO bit and sets the corresponding USERINTRAW bit (0 or 1) in the MDIO user command complete interrupt register (USERINTRAW) corresponding to USERACCESS*n* used. The corresponding USERINTMASKED bit (0 or 1) in the MDIO user command complete interrupt register (USERINTMASKED) may also be set, depending on the mask setting configured in the MDIO user command complete interrupt mask set register (USERINTMASKSET) and the MDIO user interrupt mask clear register (USERINTMASKCLEAR).

A round-robin arbitration scheme is used to schedule transactions that may be queued using both USERACCESS0 and USERACCESS1. The application software must check the status of the GO bit in USERACCESS*n* before initiating a new transaction, to ensure that the previous transaction has completed. The application software can use the ACK bit in USERACCESS*n* to determine the status of a read transaction.

2.7.2.1 Initializing the MDIO Module

The following steps are performed by the application software or device driver to initialize the MDIO device:

1. Configure the PREAMBLE and CLKDIV bits in the MDIO control register (CONTROL).
2. Enable the MDIO module by setting the ENABLE bit in CONTROL.
3. The MDIO PHY alive status register (ALIVE) can be read in polling fashion until a PHY connected to the system responded, and the MDIO PHY link status register (LINK) can determine whether this PHY already has a link.
4. Setup the appropriate PHY addresses in the MDIO user PHY select register (USERPHYSEL*n*), and set the LINKINTENB bit to enable a link change event interrupt if desirable.
5. If an interrupt on general MDIO register access is desired, set the corresponding bit in the MDIO user command complete interrupt mask set register (USERINTMASKSET) to use the MDIO user access register (USERACCESS*n*). Since only one PHY is used in this device, the application software can use one USERACCESS*n* to trigger a completion interrupt; the other USERACCESS*n* is not setup.

2.7.2.2 Writing Data To a PHY Register

The MDIO module includes a user access register (USERACCESS*n*) to directly access a specified PHY device. To write a PHY register, perform the following:

1. Check to ensure that the GO bit in the MDIO user access register (USERACCESS*n*) is cleared.
2. Write to the GO, WRITE, REGADR, PHYADR, and DATA bits in USERACCESS*n* corresponding to the PHY and PHY register you want to write.
3. The write operation to the PHY is scheduled and completed by the MDIO module. Completion of the write operation can be determined by polling the GO bit in USERACCESS*n* for a 0.
4. Completion of the operation sets the corresponding USERINTRAW bit (0 or 1) in the MDIO user command complete interrupt register (USERINTRAW) corresponding to USERACCESS*n* used. If interrupts have been enabled on this bit using the MDIO user command complete interrupt mask set register (USERINTMASKSET), then the bit is also set in the MDIO user command complete interrupt register (USERINTMASKED) and an interrupt is triggered on the CPU.

2.7.2.3 Reading Data From a PHY Register

The MDIO module includes a user access register (USERACCESS*n*) to directly access a specified PHY device. To read a PHY register, perform the following:

1. Check to ensure that the GO bit in the MDIO user access register (USERACCESS*n*) is cleared.
2. Write to the GO, REGADR, and PHYADR bits in USERACCESS*n* corresponding to the PHY and PHY register you want to read.
3. The read data value is available in the DATA bits in USERACCESS*n* after the module completes the read operation on the serial bus. Completion of the read operation can be determined by polling the GO and ACK bits in USERACCESS*n*. Once the GO bit has cleared, the ACK bit is set on a successful read.
4. Completion of the operation sets the corresponding USERINTRAW bit (0 or 1) in the MDIO user command complete interrupt register (USERINTRAW) corresponding to USERACCESS*n* used. If interrupts have been enabled on this bit using the MDIO user command complete interrupt mask set register (USERINTMASKSET), then the bit is also set in the MDIO user command complete interrupt register (USERINTMASKED) and an interrupt is triggered on the CPU.

2.7.2.4 Example of MDIO Register Access Code

The MDIO module uses the MDIO user access register (USERACCESS*n*) to access the PHY control registers. Software functions that implement the access process may simply be the following four macros:

- PHYREG_read(regadr, phyadr) — Start the process of reading a PHY register
- PHYREG_write(regadr, phyadr, data) — Start the process of writing a PHY register
- PHYREG_wait( ) — Synchronize operation (make sure read/write is idle)
- PHYREG_waitResults(results) — Wait for read to complete and return data read

Note that it is not necessary to wait after a write operation, as long as the status is checked before every operation to make sure the MDIO hardware is idle. An alternative approach is to call PHYREG_wait() after every write, and PHYREG_waitResults( ) after every read, then the hardware can be assumed to be idle when starting a new operation.

The implementation of these macros using the chip support library (CSL) is shown in Example 3 (USERACCESS0 is assumed).

Note that this implementation does not check the ACK bit in USERACCESS*n* on PHY register reads (does not follow the procedure outlined in Section 2.7.2.3). Since the MDIO PHY alive status register (ALIVE) is used to initially select a PHY, it is assumed that the PHY is acknowledging read operations. It is possible that a PHY could become inactive at a future point in time. An example of this would be a PHY that can have its MDIO addresses changed while the system is running. It is not very likely, but this condition can be tested by periodically checking the PHY state in ALIVE.

Example 3. MDIO Register Access Macros

```
#define PHYREG_read(regadr, phyadr)
        MDIO_REGS->USERACCESS0 =
             CSL_FMK(MDIO_USERACCESS0_GO,1u)                    |              /
             CSL_FMK(MDIO_USERACCESS0_REGADR,regadr)            |              /
             CSL_FMK(MDIO_USERACCESS0_PHYADR,phyadr)

#define PHYREG_write(regadr, phyadr, data)
        MDIO_REGS->USERACCESS0 =
             CSL_FMK(MDIO_USERACCESS0_GO,1u)                    |              /
             CSL_FMK(MDIO_USERACCESS0_WRITE,1)                  |              /
             CSL_FMK(MDIO_USERACCESS0_REGADR,regadr)            |              /
             CSL_FMK(MDIO_USERACCESS0_PHYADR,phyadr)            |              /

             CSL_FMK(MDIO_USERACCESS0_DATA, data)

#define PHYREG_wait()
        while( CSL_FEXT(MDIO_REGS->USERACCESS0,MDIO_USERACCESS0_GO) )

#define PHYREG_waitResults( results ) {
        while( CSL_FEXT(MDIO_REGS->USERACCESS0,MDIO_USERACCESS0_GO) );
        results = CSL_FEXT(MDIO_REGS->USERACCESS0, MDIO_USERACCESS0_DATA); }
```

2.8 EMAC Module

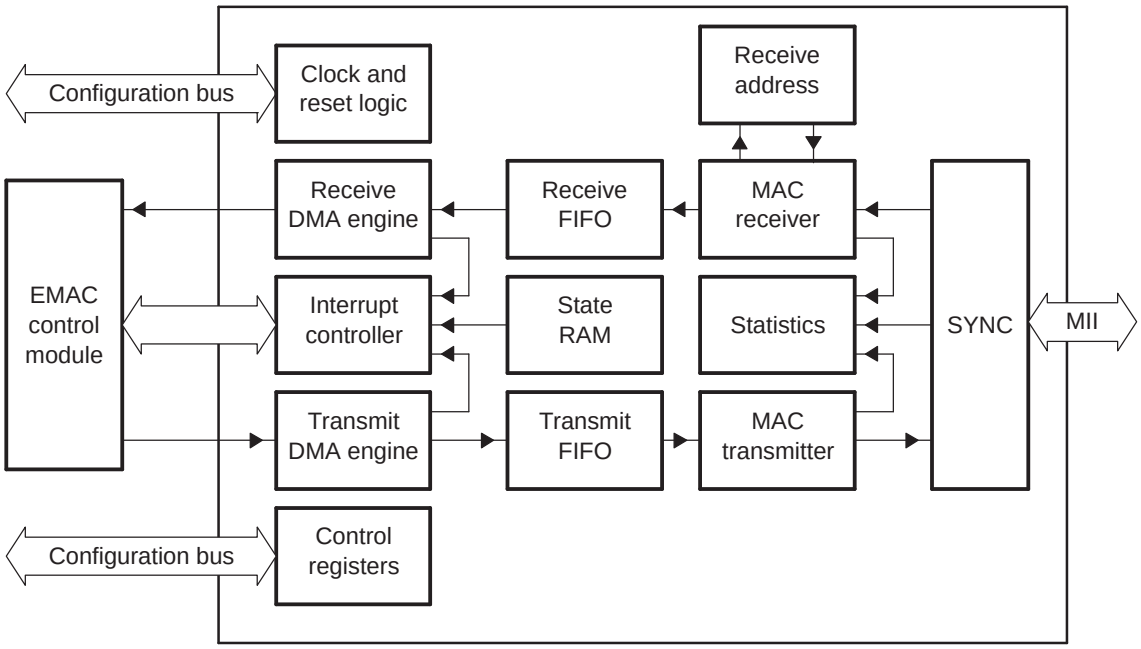
This section discusses the architecture and basic function of the EMAC module.

2.8.1 EMAC Module Components

The EMAC module (Figure 10) interfaces to the outside world through the Media Independent Interface (MII) and interfaces to the system core through the EMAC control module. The EMAC consists of the following logical components:

- The receive path includes: receive DMA engine, receive FIFO, and MAC receiver
- The transmit path includes: transmit DMA engine, transmit FIFO, and MAC transmitter
- Statistics logic
- State RAM
- Interrupt controller
- Control registers and logic
- Clock and reset logic

Figure 10. EMAC Module Block Diagram



2.8.1.1 Receive DMA Engine

The receive DMA engine is the interface between the receive FIFO and the system core. It interfaces to the CPU through the bus arbiter in the EMAC control module. This DMA engine is totally independent of the device DMA.

2.8.1.2 Receive FIFO

The receive FIFO consists of three cells of 64-bytes each and associated control logic. The FIFO buffers receive data in preparation for writing into packet buffers in device memory.

2.8.1.3 MAC Receiver

The MAC receiver detects and processes incoming network frames, de-frames them, and puts them into the receive FIFO. The MAC receiver also detects errors and passes statistics to the statistics RAM.

2.8.1.4 Transmit DMA Engine

The transmit DMA engine is the interface between the transmit FIFO and the CPU. It interfaces to the CPU through the bus arbiter in the EMAC control module.

2.8.1.5 Transmit FIFO

The transmit FIFO consists of three cells of 64–bytes each and associated control logic. The FIFO buffers data in preparation for transmission.

2.8.1.6 MAC Transmitter

The MAC transmitter formats frame data from the transmit FIFO and transmits the data using the CSMA/CD access protocol. The frame CRC can be automatically appended, if required. The MAC transmitter also detects transmission errors and passes statistics to the statistics registers.

2.8.1.7 Statistics Logic

The Ethernet statistics are counted and stored in the statistics logic RAM. This statistics RAM keeps track of 36 different Ethernet packet statistics.

2.8.1.8 State RAM

State RAM contains the head descriptor pointers and completion pointers registers for both transmit and receive channels.

2.8.1.9 EMAC Interrupt Controller

The interrupt controller contains the interrupt related registers and logic. The 18 raw EMAC interrupts are input to this submodule and masked module interrupts are output.

2.8.1.10 Control Registers and Logic

The EMAC is controlled by a set of memory-mapped registers. The control logic also signals transmit, receive, and status related interrupts to the CPU through the EMAC control module.

2.8.1.11 Clock and Reset Logic

The clock and reset submodule generates all the EMAC clocks and resets. For more details on reset capabilities, see Section 2.14.1.

2.8.2 EMAC Module Operational Overview

After reset, initialization, and configuration, the host may initiate transmit operations. Transmit operations are initiated by host writes to the appropriate transmit channel head descriptor pointer contained in the state RAM block. The transmit DMA controller then fetches the first packet in the packet chain from memory. The DMA controller writes the packet into the transmit FIFO in bursts of 64-byte cells. When the threshold number of cells, configurable using the TXCELLTHRESH bit in the FIFO control register (FIFOCONTROL), have been written to the transmit FIFO, or a complete packet, whichever is smaller, the MAC transmitter then initiates the packet transmission. The SYNC block transmits the packet over the MII interfaces in accordance with the 802.3 protocol. Transmit statistics are counted by the statistics block.

Receive operations are initiated by host writes to the appropriate receive channel head descriptor pointer after host initialization and configuration. The SYNC submodule receives packets and strips off the Ethernet related protocol. The packet data is input to the MAC receiver, which checks for address match and processes errors. Accepted packets are then written to the receive FIFO in bursts of 64-byte cells. The receive DMA controller then writes the packet data to memory. Receive statistics are counted by the statistics block.

The EMAC module operates independently of the CPU. It is configured and controlled by its register set mapped into device memory. Information about data packets is communicated by use of 16-byte descriptors that are placed in an 8K-byte block of RAM in the EMAC control module.

For transmit operations, each 16-byte descriptor describes a packet or packet fragment in the system's internal or external memory. For receive operations, each 16-byte descriptor represents a free packet buffer or buffer fragment. On both transmit and receive, an Ethernet packet is allowed to span one or more memory fragments, represented by one 16-byte descriptor per fragment. In typical operation, there is only one descriptor per receive buffer, but transmit packets may be fragmented, depending on the software architecture.

An interrupt is issued to the CPU whenever a transmit or receive operation has completed. However, it is not necessary for the CPU to service the interrupt while there are additional resources available. In other words, the EMAC continues to receive Ethernet packets until its receive descriptor list has been exhausted. On transmit operations, the transmit descriptors need only be serviced to recover their associated memory buffer. Thus, it is possible to delay servicing of the EMAC interrupt if there are real-time tasks to perform.

Eight channels are supplied for both transmit and receive operations. On transmit, the eight channels represent eight independent transmit queues. The EMAC can be configured to treat these channels as an equal priority "round-robin" queue or as a set of eight fixed-priority queues. On receive, the eight channels represent eight independent receive queues with packet classification. Packets are classified based on the destination MAC address. Each of the eight channels is assigned its own MAC address, enabling the EMAC module to act like eight virtual MAC adapters. Also, specific types of frames can be sent to specific channels. For example, multicast, broadcast, or other (promiscuous, error, etc.), can each be received on a specific receive channel queue.

The EMAC keeps track of 36 different statistics, plus keeps the status of each individual packet in its corresponding packet descriptor.

2.9 Media Independent Interface (MII)

The following sections discuss the operation of the Media Independent Interface (MII) in 10 Mbps and 100 Mbps mode. An IEEE 802.3 compliant Ethernet MAC controls the interface.

2.9.1 Data Reception

2.9.1.1 Receive Control

Data received from the PHY is interpreted and output to the EMAC receive FIFO. Interpretation involves detection and removal of the preamble and start-of-frame delimiter, extraction of the address and frame length, data handling, error checking and reporting, cyclic redundancy checking (CRC), and statistics control signal generation. Address detection and frame filtering is performed outside the MII interface.

2.9.1.2 Receive Inter-Frame Interval

The 802.3 standard requires an interpacket gap (IPG), which is 24 MII clocks (96 bit times). However, the EMAC can tolerate a reduced IPG (2 MII clocks or 8 bit times) with a correct preamble and start frame delimiter. This interval between frames must comprise (in the following order):

1. An Interpacket Gap (IPG).
2. A 7-byte preamble (all bytes 55h).
3. A 1-byte start of frame delimiter (5DH).

2.9.1.3 Receive Flow Control

When enabled and triggered, receive flow control is initiated to limit the EMAC from further frame reception. Two forms of receive buffer flow control are available:

- Collision-based flow control for half-duplex mode
- IEEE 802.3x pause frames flow control for full-duplex mode

In either case, receive flow control prevents frame reception by issuing the flow control appropriate for the current mode of operation. Receive flow control prevents reception of frames on the EMAC until all of the triggering conditions clear, at which time frames may again be received by the EMAC.

Receive flow control is enabled by the RXBUFFERFLOWEN bit in the MAC control register (MACCONTROL). The EMAC is configured for collision or IEEE 802.3X flow control using the FULLDUPLEX bit in MACCONTROL. Receive flow control is triggered when the number of free buffers in any enabled receive channel free buffer count register (RX*n*FREEBUFFER) is less than or equal to the receive channel flow control threshold register (RX*n*FLOWTHRESH) value. Receive flow control is independent of receive QOS, except that both use the free buffer values.

2.9.1.3.1 Collision-Based Receive Buffer Flow Control

Collision-based receive buffer flow control provides a means of preventing frame reception when the EMAC is operating in half-duplex mode (the FULLDUPLEX bit is cleared in MACCONTROL). When receive flow control is enabled and triggered, the EMAC generates collisions for received frames. The jam sequence transmitted is the 12-byte sequence C3.C3.C3.C3.C3.C3.C3.C3.C3.C3.C3.C3h. The jam sequence begins no later than approximately as the source address starts to be received. Note that these forced collisions are not limited to a maximum of 16 consecutive collisions, and are independent of the normal back-off algorithm.

Receive flow control does not depend on the value of the incoming frame destination address. A collision is generated for any incoming packet, regardless of the destination address, if any EMAC enabled channel's free buffer register value is less than or equal to the channel's flow threshold value.

2.9.1.3.2 IEEE 802.3x-Based Receive Buffer Flow Control

IEEE 802.3x-based receive buffer flow control provides a means of preventing frame reception when the EMAC is operating in full-duplex mode (the FULLDUPLEX bit is set in MACCONTROL). When receive flow control is enabled and triggered, the EMAC transmits a pause frame to request that the sending station stop transmitting for the period indicated within the transmitted pause frame.

The EMAC transmits a pause frame to the reserved multicast address at the first available opportunity (immediately if currently idle or following the completion of the frame currently being transmitted). The pause frame contains the maximum possible value for the pause time (FFFFh). The EMAC counts the receive pause frame time (decrements FF00h to 0) and retransmits an outgoing pause frame, if the count reaches 0. When the flow control request is removed, the EMAC transmits a pause frame with a zero pause time to cancel the pause request.

Note that transmitted pause frames are only a request to the other end station to stop transmitting. Frames that are received during the pause interval are received normally (provided the receive FIFO is not full).

Pause frames are transmitted if enabled and triggered, regardless of whether or not the EMAC is observing the pause time period from an incoming pause frame.

The EMAC transmits pause frames as described below:

- The 48-bit reserved multicast destination address 01.80.C2.00.00.01h.
- The 48-bit source address (set using the MACSRCADDRLO and MACSRCADDRHI registers).
- The 16-bit length/type field containing the value 88.08h.
- The 16-bit pause opcode equal to 00.01h.
- The 16-bit pause time value of FF.FFh. A pause-quantum is 512 bit-times. Pause frames sent to cancel a pause request have a pause time value of 00.00h.
- Zero padding to 64-byte data length (EMAC transmits only 64-byte pause frames).
- The 32-bit frame-check sequence (CRC word).

All quantities are hexadecimal and are transmitted most-significant byte first. The least-significant bit (LSB) is transferred first in each byte.

If the RXBUFFERFLOWEN bit in MACCONTROL is cleared to 0 while the pause time is nonzero, then the pause time is cleared to 0 and a zero count pause frame is sent.

2.9.2 Data Transmission

The EMAC passes data to the PHY from the transmit FIFO (when enabled). Data is synchronized to the transmit clock rate. Transmission begins when there are TXCELLTHRESH cells of 64 bytes each, or a complete packet, in the FIFO.

2.9.2.1 Transmit Control

A jam sequence is output if a collision is detected on a transmit packet. If the collision was late (after the first 64 bytes have been transmitted), the collision is ignored. If the collision is not late, the controller will back off before retrying the frame transmission. When operating in full-duplex mode, the carrier sense (MCRS) and collision-sensing (MCOL) modes are disabled.

2.9.2.2 CRC Insertion

If the SOP buffer descriptor PASSCRC flag is cleared, the EMAC generates and appends a 32-bit Ethernet CRC onto the transmitted data. For the EMAC-generated CRC case, a CRC (or placeholder) at the end of the data is allowed but not required. The buffer byte count value should not include the CRC bytes, if they are present.

If the SOP buffer descriptor PASSCRC flag is set, then the last four bytes of the transmit data are transmitted as the frame CRC. The four CRC data bytes should be the last four bytes of the frame and should be included in the buffer byte count value. The MAC performs no error checking on the outgoing CRC.

2.9.2.3 Adaptive Performance Optimization (APO)

The EMAC incorporates adaptive performance optimization (APO) logic that may be enabled by setting the TXPACE bit in the MAC control register (MACCONTROL). Transmission pacing to enhance performance is enabled when the TXPACE bit is set. Adaptive performance pacing introduces delays into the normal transmission of frames, delaying transmission attempts between stations, reducing the probability of collisions occurring during heavy traffic (as indicated by frame deferrals and collisions), thereby, increasing the chance of successful transmission.

When a frame is deferred, suffers a single collision, multiple collisions, or excessive collisions, the pacing counter is loaded with an initial value of 31. When a frame is transmitted successfully (without experiencing a deferral, single collision, multiple collision, or excessive collision), the pacing counter is decremented by 1, down to 0.

With pacing enabled, a new frame is permitted to immediately (after one interpacket gap) attempt transmission only if the pacing counter is 0. If the pacing counter is nonzero, the frame is delayed by the pacing delay of approximately four interpacket gap (IPG)delays. APO only affects the IPG preceding the first attempt at transmitting a frame; APO does not affect the back-off algorithm for retransmitted frames.

2.9.2.4 Interpacket-Gap (IPG) Enforcement

The measurement reference for the IPG of 96 bit times is changed depending on frame traffic conditions. If a frame is successfully transmitted without collision and MCRS is deasserted within approximately 48 bit times of MTXEN being deasserted, then 96 bit times is measured from MTXEN. If the frame suffered a collision or MCRS is not deasserted until more than approximately 48 bit times after MTXEN is deasserted, then 96 bit times (approximately, but not less) is measured from MCRS.

2.9.2.5 Back Off

The EMAC implements the 802.3 binary exponential back-off algorithm.

2.9.2.6 Transmit Flow Control

Incoming pause frames are acted upon, when enabled, to prevent the EMAC from transmitting any further frames. Incoming pause frames are only acted upon when the FULLDUPLEX and TXFLOWEN bits in the MAC control register (MACCONTROL) are set. Pause frames are not acted upon in half-duplex mode. Pause frame action is taken if enabled, but normally the frame is filtered and not transferred to memory. MAC control frames are transferred to memory, if the RXCMFEN bit in the receive multicast/broadcast/promiscuous channel enable register (RXMBPENABLE) is set. The TXFLOWEN and FULLDUPLEX bits affect whether or not MAC control frames are acted upon, but they have no affect upon whether or not MAC control frames are transferred to memory or filtered.

Pause frames are a subset of MAC control frames with an opcode field of 0001h. Incoming pause frames are only acted upon by the EMAC if:

- TXFLOWEN bit is set in M ACCONTROL
- The frame's length is 64 to RXMAXLEN bytes inclusive
- The frame contains no CRC error or align/code errors

The pause time value from valid frames is extracted from the two bytes following the opcode. The pause time is loaded into the EMAC transmit pause timer and the transmit pause time period begins. If a valid pause frame is received during the transmit pause time period of a previous transmit pause frame then:

- If the destination address is not equal to the reserved multicast address or any enabled or disabled unicast address, then the transmit pause timer immediately expires, or
- If the new pause time value is 0, then the transmit pause timer immediately expires, else
- The EMAC transmit pause timer immediately is set to the new pause frame pause time value. (Any remaining pause time from the previous pause frame is discarded).

If the TXFLOWEN bit in MACCONTROL is cleared, then the pause timer immediately expires.

The EMAC does not start the transmission of a new data frame any sooner than 512 bit-times after a pause frame with a nonzero pause time has finished being received (MRXDV going inactive). No transmission begins until the pause timer has expired (the EMAC may transmit pause frames in order to initiate outgoing flow control). Any frame already in transmission when a pause frame is received is completed and unaffected.

Incoming pause frames consist of:

- A 48-bit destination address equal to one of the following:
  - The reserved multicast destination address 01.80.C2.00.00.01h
  - Any EMAC 48-bit unicast address. Pause frames are accepted, regardless of whether the channel is enabled or not.
- The 16-bit length/type field containing the value 88.08h.
- The 48-bit source address of the transmitting device.
- The 16-bit pause opcode equal to 00.01h.
- The 16-bit pause time. A pause-quantum is 512 bit-times.
- Padding to 64-byte data length.
- The 32-bit frame-check sequence (CRC word).

All quantities are hexadecimal and are transmitted most-significant byte first. The least-significant bit (LSB) is transferred first in each byte.

The padding is required to make up the frame to a minimum of 64 bytes. The standard allows pause frames longer than 64 bytes to be discarded or interpreted as valid pause frames. The EMAC recognizes any pause frame between 64 bytes and RXMAXLEN bytes in length.

2.9.2.7 Speed, Duplex, and Pause Frame Support

The MAC operates at 10 Mbps or 100 Mbps, in half-duplex or full-duplex mode, and with or without pause frame support as configured by the host.

2.10 Packet Receive Operation

2.10.1 Receive DMA Host Configuration

To configure the receive DMA for operation the host must:

- Initialize the receive addresses.
- Initialize the receive channel *n* DMA head descriptor pointer registers (RX*n*HDP) to 0.
- Write the MAC address hash *n* registers (MACHASH1 and MACHASH2), if multicast addressing is desired.
- If flow control is to be enabled, initialize:
  - the receive channel *n* free buffer count registers (RX*n*FREEBUFFER)
  - the receive channel *n* flow control threshold register (RX*n*FLOWTHRESH)
  - the receive filter low priority frame threshold register (RXFILTERLOWTHRESH)
- Enable the desired receive interrupts using the receive interrupt mask set register (RXINTMASKSET) and the receive interrupt mask clear register (RXINTMASKCLEAR).
- Set the appropriate configuration bits in the MAC control register (MACCONTROL).
- Write the receive buffer offset register (RXBUFFEROFFSET) value (typically zero).
- Setup the receive channel(s) buffer descriptors and initialize RX*n*HDP.
- Enable the receive DMA controller by setting the RXEN bit in the receive control register (RXCONTROL).
- Configure and enable the receive operation, as desired, in the receive multicast/broadcast/promiscuous channel enable register (RXMBPENABLE) and by using the receive unicast set register (RXUNICASTSET) and the receive unicast clear register (RXUNICASTCLEAR).

2.10.2 Receive Channel Enabling

Each of the eight receive channels has an enable bit (RXCH*n*EN) in the receive unicast set register (RXUNICASTSET) that is controlled using RXUNICASTSET and the receive unicast clear register (RXUNICASTCLEAR). The RXCH*n*EN bits determine whether the given channel is enabled (when set to 1) to receive frames with a matching unicast or multicast destination address.

The RXBROADEN bit in the receive multicast/broadcast/promiscuous channel enable register (RXMBPENABLE) determines if broadcast frames are enabled or filtered. If broadcast frames are enabled (when set to 1), then they are copied to only a single channel selected by the RXBROADCH bit in RXMBPENABLE.

The RXMULTEN bit in RXMBPENABLE determines if hash matching multicast frames are enabled or filtered. Incoming multicast addresses (group addresses) are hashed into an index in the hash table. If the indexed bit is set then the frame hash matches and will be transferred to the channel selected by the RXMULTCH bit in RXMBPENABLE when multicast frames are enabled. The multicast hash bits are set in the MAC address hash *n* registers (MACHASH1 and MACHASH2).

The RXPROMCH bit in RXMBPENABLE selects the promiscuous channel to receive frames selected by the RXCMFEN, RXCSFEN, RXCEFEN, and RXCAFEN bits. These four bits allow reception of MAC control frames, short frames, error frames, and all frames (promiscuous), respectively.

2.10.3 Receive Address Matching

All eight MAC addresses corresponding to the eight receive channels share the upper 40 bits. Only the lower byte is unique for each address. All eight receive addresses should be initialized, because pause frames are acted upon regardless of whether a channel is enabled or not.

A MAC address is written by first writing the address number (channel) to be written into the MAC index register (MACINDEX). The upper 32 bits of address are then written to the MAC address high bytes register (MACADDRHI), which is followed by writing the lower 16 bits of address to the MAC address low bytes register (MACADDRLO). Since all eight MAC addresses share the upper 40 bits of address, MACADDRHI needs to be written only the first time (for the first channel configured).

2.10.4 Hardware Receive QOS Support

Hardware receive quality of service (QOS) is supported, when enabled, by the Tag Protocol Identifier format and the associated Tag Control Information (TCI) format priority field. When the incoming frame length/type value is equal to 81.00h, the EMAC recognizes the frame as an Ethernet Encoded Tag Protocol Type. The two octets immediately following the protocol type contain the 16-bit TCI field. Bits 15-13 of the TCI field contain the received frames priority (0 to 7). The received frame is a low-priority frame, if the priority value is 0 to 3; the received frame is a high-priority frame, if the priority value is 4 to 7. All frames that have a length/type field value not equal to 81.00h are low-priority frames. Received frames that contain priority information are determined by the EMAC as:

- A 48-bit (6 bytes) destination address equal to:
  - The destination station's individual unicast address.
  - The destination station's multicast address (MACHASH1 and MACHASH2).
  - The broadcast address of all ones.
- A 48-byte (6 bytes) source address.
- The 16-bit (2 bytes) length/type field containing the value 81.00h.
- The 16-bit (2 bytes) TCI field with the priority field in the upper 3 bits.
- Data bytes
- The 4 bytes CRC.

The receive filter low priority frame threshold register (RXFILTERLOWTHRESH) and the receive channel *n* free buffer count registers (RX*n*FREEBUFFER) are used in conjunction with the priority information to implement receive hardware QOS. Low-priority frames are filtered if the number of free buffers (RX*n*FREEBUFFER) for the frame channel is less than or equal to the filter low threshold (RXFILTERLOWTHRESH) value. Hardware QOS is enabled by the RXQOSEN bit in the receive multicast/broadcast/promiscuous channel enable register (RXMBPENABLE).

2.10.5 Host Free Buffer Tracking

The host must track free buffers for each enabled channel (including unicast, multicast, broadcast, and promiscuous), if receive QOS or receive flow control is used. Disabled channel free buffer values are do not cares. During initialization, the host should write the number of free buffers for each enabled channel to the appropriate receive channel *n* free buffer count registers (RX*n*FREEBUFFER). The EMAC decrements the appropriate channel's free buffer value for each buffer used. When the host reclaims the frame buffers, the host should write the channel free buffer register with the number of reclaimed buffers (write to increment). There are a maximum of 65,535 free buffers available. RX*n*FREEBUFFER only needs to be updated by the host if receive QOS or flow control is used.

2.10.6 Receive Channel Teardown

The host commands a receive channel teardown by writing the channel number to the receive teardown register (RXTEARDOWN). When a teardown command is issued to an enabled receive channel, the following occurs:

- Any current frame in reception completes normally.
- The TDOWNCMPLT flag is set in the next buffer descriptor in the chain, if there is one.
- The channel head descriptor pointer is cleared to 0.
- A receive interrupt for the channel is issued to the host.
- The corresponding receive channel *n* completion pointer register (RX*n*CP) contains the value FFFF FFCh.

Channel teardown may be commanded on any channel at any time. The host is informed of the teardown completion by the set teardown complete (TDOWNCMPLT) buffer descriptor bit. The EMAC does not clear any channel enables due to a teardown command. A teardown command to an inactive channel issues an interrupt that software should acknowledge with an FFFF FFFCh acknowledge value to RX*n*CP (note that there is no buffer descriptor in this case). Software may read RX*n*CP to determine if the interrupt was due to a commanded teardown. The read value is FFFF FFFCh, if the interrupt was due to a teardown command.

2.10.7 Receive Frame Classification

Received frames are proper (good) frames, if they are between 64 bytes and the value in the receive maximum length register (RXMAXLEN) bytes in length (inclusive) and contain no code, align, or CRC errors.

Received frames are long frames, if their frame count exceeds the value in RXMAXLEN. The RXMAXLEN reset (default) value is 5EEh (1518 in decimal). Long received frames are either oversized or jabber frames. Long frames with no errors are oversized frames; long frames with CRC, code, or alignment errors are jabber frames.

Received frames are short frames, if their frame count is less than 64 bytes. Short frames that address match and contain no errors are undersized frames; short frames with CRC, code, or alignment errors are fragment frames. If the frame length is less than or equal to 20, then the frame CRC is passed, regardless of whether the RXPASSCRC bit is set or cleared in the receive multicast/broadcast/promiscuous channel enable register (RXMBPENABLE).

A received long packet always contains RXMAXLEN number of bytes transferred to memory (if the RXCEFEN bit is set in RXMBPENABLE), regardless of the value of the RXPASSCRC bit. Following is an example with RXMAXLEN set to 1518:

- If the frame length is 1518, then the packet is not a long packet and there are 1514 or 1518 bytes transferred to memory depending on the value of the RXPASSCRC bit.
- If the frame length is 1519, there are 1518 bytes transferred to memory regardless of the RXPASSCRC bit value. The last three bytes are the first three CRC bytes.
- If the frame length is 1520, there are 1518 bytes transferred to memory regardless of the RXPASSCRC bit value. The last two bytes are the first two CRC bytes.
- If the frame length is 1521, there are 1518 bytes transferred to memory regardless of the RXPASSCRC bit value. The last byte is the first CRC byte.
- If the frame length is 1522, there are 1518 bytes transferred to memory. The last byte is the last data byte.

2.10.8 Promiscuous Receive Mode

When the promiscuous receive mode is enabled by setting the RXCAFEN bit in the receive multicast/broadcast/promiscuous channel enable register (RXMBPENABLE), nonaddress matching frames that would normally be filtered are transferred to the promiscuous channel. Address matching frames that would normally be filtered due to errors are transferred to the address match channel when the RXCAFEN and RXCEFEN bits in RXMBPENABLE are set. A frame is considered to be an address matching frame only if it is enabled to be received on a unicast, multicast, or broadcast channel. Frames received to disabled unicast, multicast, or broadcast channels are considered nonaddress matching.

MAC control frames address match only if the RXCMFEN bit in RXMBPENABLE is set. The RXCEFEN and RXCSFEN bits in RXMBPENABLE determine whether error frames are transferred to memory or not, but they do not determine whether error frames are address matching or not. Short frames are a special type of error frames.

A single channel is selected as the promiscuous channel by the RXPROMCH bit in RXMBPENABLE. The promiscuous receive mode is enabled by the RXCMFEN, RXCEFEN, RXCSFEN, and RXCAFEN bits in RXMBPENABLE. Table 4 shows the effects of the promiscuous enable bits. Proper frames are frames that are between 64 bytes and the value in the receive maximum length register (RXMAXLEN) bytes in length inclusive and contain no code, align, or CRC errors.

Table 4. Receive Frame Treatment Summary

Address Match	RXCAFEN	RXCEFEN	RXCMFEN	RXCSFEN	Receive Frame Treatment
0	0	X	X	X	No frames transferred.
0	1	0	0	0	Proper frames transferred to promiscuous channel.
0	1	0	0	1	Proper/undersized data frames transferred to promiscuous channel.
0	1	0	1	0	Proper data and control frames transferred to promiscuous channel.
0	1	0	1	1	Proper/undersized data and control frames transferred to promiscuous channel.
0	1	1	0	0	Proper/oversize/jabber/code/align/CRC data frames transferred to promiscuous channel. No control or undersized/fragment frames are transferred.
0	1	1	0	1	Proper/undersized/fragment/oversize/jabber/code/ align/CRC data frames transferred to promiscuous channel. No control frames are transferred.
0	1	1	1	0	Proper/oversize/jabber/code/align/CRC data and control frames transferred to promiscuous channel. No undersized frames are transferred.
0	1	1	1	1	All nonaddress matching frames with and without errors transferred to promiscuous channel.
1	X	0	0	0	Proper data frames transferred to address match channel.
1	X	0	0	1	Proper/undersized data frames transferred to address match channel.
1	X	0	1	0	Proper data and control frames transferred to address match channel.
1	X	0	1	1	Proper/undersized data and control frames transferred to address match channel.
1	X	1	0	0	Proper/oversize/jabber/code/align/CRC data frames transferred to address match channel. No control or undersized frames are transferred.
1	X	1	0	1	Proper/oversize/jabber/fragment/undersized/code/ align/CRC data frames transferred to address match channel. No control frames are transferred.
1	X	1	1	0	Proper/oversize/jabber/code/align/CRC data and control frames transferred to address match channel. No undersized/fragment frames are transferred.
1	X	1	1	1	All address matching frames with and without errors transferred to the address match channel

2.10.9 Receive Overrun

The types of receive overrun are:

- FIFO start of frame overrun (FIFO_SOF)
- FIFO middle of frame overrun (FIFO_MOF)
- DMA start of frame overrun (DMA_SOF)
- DMA middle of frame overrun (DMA_MOF)

The statistics counters used to track these types of receive overrun are:

- Receive start of frame overruns register (RXSOFOVERRUNS)
- Receive middle of frame overruns register (RXMOFOVERRUNS)
- Receive DMA overruns register (RXDMAOVERRUNS)

Start of frame overruns happen when there are no resources available when frame reception begins. Start of frame overruns increment the appropriate overrun statistic(s) and the frame is filtered.

Middle of frame overruns happen when there are some resources to start the frame reception, but the resources run out during frame reception. In normal operation, a frame that overruns after starting the frame reception is filtered and the appropriate statistic(s) are incremented; however, the RXCEFEN bit in the receive multicast/broadcast/promiscuous channel enable register (RXMBPENABLE) affects overrun frame treatment. Table 5 shows how the overrun condition is handled for the middle of frame overrun.

Table 5. Middle of Frame Overrun Treatment

Address Match	RXCAFEN	RXCEFEN	Middle of Frame Overrun Treatment
0	0	X	Overrun frame filtered.
0	1	0	Overrun frame filtered.
0	1	1	As much frame data as possible is transferred to the promiscuous channel until overrun. The appropriate overrun statistic(s) is incremented and the OVERRUN and NOMATCH flags are set in the SOP buffer descriptor. Note that the RXMAXLEN number of bytes cannot be reached for an overrun to occur (it would be truncated and be a jabber or oversize).
1	X	0	Overrun frame filtered with the appropriate overrun statistic(s) incremented.
1	X	1	As much frame data as possible is transferred to the address match channel until overrun. The appropriate overrun statistic(s) is incremented and the OVERRUN flag is set in the SOP buffer descriptor. Note that the RXMAXLEN number of bytes cannot be reached for an overrun to occur (it would be truncated).

2.11 Packet Transmit Operation

The transmit DMA is an eight channel interface. Priority between the eight queues may be either fixed or round-robin as selected by the TXPTYPE bit in the MAC control register (MACCONTROL). If the priority type is fixed, then channel 7 has the highest priority and channel 0 has the lowest priority. Round-robin priority proceeds from channel 0 to channel 7.

2.11.1 Transmit DMA Host Configuration

To configure the transmit DMA for operation the host must perform:

- Write the MAC source address low bytes register (MACSRCADDRLO) and the MAC source address high bytes register (MACSRCADDRHI) (used for pause frames on transmit).
- Initialize the transmit channel *n* DMA head descriptor pointer registers (TX*n*HDP) to 0.
- Enable the desired transmit interrupts using the transmit interrupt mask set register (TXINTMASKSET) and the transmit interrupt mask clear register (TXINTMASKCLEAR).
- Set the appropriate configuration bits in the MAC control register (MACCONTROL).
- Setup the transmit channel(s) buffer descriptors in host memory.
- Enable the transmit DMA controller by setting the TXEN bit in the transmit control register (TXCONTROL).
- Write the appropriate TX*n*HDP with the pointer to the first descriptor to start transmit operations.

2.11.2 Transmit Channel Teardown

The host commands a transmit channel teardown by writing the channel number to the transmit teardown register (TXTEARDOWN). When a teardown command is issued to an enabled transmit channel, the following occurs:

- Any frame currently in transmission completes normally.
- The TDOWNCMPLT flag is set in the next SOP buffer descriptor in the chain, if there is one.
- The channel head descriptor pointer is cleared to 0.
- A transmit interrupt is issued to inform the host of the channel teardown.
- The corresponding transmit channel *n* completion pointer register (TX*n*CP) contains the value FFFF FFFCh.
- The host should acknowledge a teardown interrupt with an FFFF FFFCh acknowledge value.

Channel teardown may be commanded on any channel at any time. The host is informed of the teardown completion by the set teardown complete (TDOWNCMPLT) buffer descriptor bit. The EMAC does not clear any channel enables due to a teardown command. A teardown command to an inactive channel issues an interrupt that software should acknowledge with an FFFF FFFCh acknowledge value to TX*n*CP (note that there is no buffer descriptor in this case). Software may read the interrupt acknowledge location (TX*n*CP) to determine if the interrupt was due to a commanded teardown. The read value is FFFF FFFCh, if the interrupt was due to a teardown command.

2.12 Receive and Transmit Latency

The transmit and receive FIFOs each contain three 64-byte cells. The EMAC begins transmission of a packet on the wire after TXCELLTHRESH (configurable through the FIFO control register) cells, or a complete packet, are available in the FIFO.

Transmit underrun cannot occur for packet sizes of TXCELLTHRESH times 64 bytes (or less). For larger packet sizes, transmit underrun occurs if the memory latency is greater than the time required to transmit a 64-byte cell on the wire; this is 5.12 µs in 100 Mbps mode and 51.2 µs in 10 Mbps mode. The memory latency time includes all buffer descriptor reads for the entire cell data.

Receive overrun is prevented if the receive memory cell latency is less than the time required to transmit a 64-byte cell on the wire: 5.12 µs in 100 Mbps mode, or 51.2 µs in 10 Mbps mode. The latency time includes any required buffer descriptor reads for the cell data.

Latency to system's internal and external RAM can be controlled through the use of the transfer node priority allocation register available at the device level. Latency to descriptor RAM is low because RAM is local to the EMAC, as it is part of the EMAC control module.

2.13 Transfer Node Priority

The DM643x device contains a chip-level register, master priority register (MSTPRI), that is used to set the priority of the transfer node used in issuing memory transfer requests to system memory.

Although the EMAC has internal FIFOs to help alleviate memory transfer arbitration problems, the average transfer rate of data read and written by the EMAC to internal or external processor memory must be at least that of the Ethernet wire rate. In addition, the internal FIFO system can not withstand a single memory latency event greater than the time it takes to fill or empty a TXCELLTHRESH number of internal 64 byte FIFO cells.

For 100 Mbps operation, these restrictions translate into the following rules:

- The short-term average, each 64-byte memory read/write request from the EMAC must be serviced in no more than 5.12 µs.
- Any single latency event in request servicing can be no longer than (5.12 × TXCELLTHRESH) µs.

Bits [0-2] of the second chip-level master priority register (MSTPRI1) are used to set the transfer node priority within the Switched Central Resource (SCR5) for the EMAC master peripheral.

A value of 000b has the highest priority, while 111b has the lowest priority. The default priority assigned to the EMAC is 100b. It is important to have a balance between all peripherals. In most cases, the default priorities will not need adjustment. For more information on the master peripherals priorities, see the device-specific data manual.

2.14 Reset Considerations

2.14.1 Software Reset Considerations

Peripheral clock and reset control is done through the Power and Sleep Controller (PSC) module included with the device. For more on how the EMAC, MDIO, and EMAC control module are disabled or placed in reset at runtime from the registers located in the PSC module, see Section 2.17.

With the EMAC still in reset (PSC in the default state):

1. Program the PINMUX1 register to HOSTBK = 3h or 4h (MII).
2. Program the VDD3P3V_PWDN register to power up the IO pins for MII pins (see the device-specific data manual).
3. Program the PSC to enale the EMAC. For information on how to enable the EMAC peripheral from the PSC, see the *TMS320DM643x DMP DSP Subsystem Reference Guide* (SPRU978).

Within the peripheral itself, the EMAC component of the Ethernet MAC peripheral can be placed in a reset state by writing to the soft reset register (SOFTRESET). Writing a 1 to the SOFTRESET bit, causes the EMAC logic to be reset and the register values to be set to their default values. Software reset occurs when the receive and transmit DMA controllers are in an idle state to avoid locking up the configuration bus; it is the responsibility of the software to verify that there are no pending frames to be transferred. After writing a 1 to the SOFTRESET bit, it may be polled to determine if the reset has occurred. If a 1 is read, the reset has not yet occurred; if a 0 is read, then a reset has occurred.

After a software reset operation, all the EMAC registers need to be reinitialized for proper data transmission, including the FULLDUPLEX bit setting in the MAC control register (MACCONTROL).

Unlike the EMAC module, the MDIO and EMAC control modules cannot be placed in reset from a register inside their memory map.

2.14.2 Hardware Reset Considerations

When a hardware reset occurs, the EMAC peripheral has its register values reset and all the components return to their default state. After the hardware reset, the EMAC needs to be initialized before being able to resume its data transmission, as described in Section 2.15.

A hardware reset is the only means of recovering from the error interrupts (HOSTPEND), which are triggered by errors in packet buffer descriptors. Before doing a hardware reset, you should inspect the error codes in the MAC status register (MACSTATUS) that gives information about the type of software error that needs to be corrected. For detailed information on error interrupts, see Section 2.16.1.4.

2.15 Initialization

2.15.1 Enabling the EMAC/MDIO Peripheral

When the device is powered on, the EMAC peripheral is in a disabled state. Before any EMAC specific initialization can take place, the EMAC needs to be enabled; otherwise, its registers cannot be written and the reads will all return a value of zero.

The EMAC/MDIO is enabled through the Power and Sleep Controller (PSC) registers. For information on how to enable the EMAC peripheral from the Power and Sleep Controller, see the *TMS320DM643x DMP DSP Subsystem Reference Guide* (SPRU978).

When first enabled, the EMAC peripheral registers are set to their default values. After enabling the peripheral, you may proceed with the module specific initialization.

2.15.2 EMAC Control Module Initialization

The EMAC control module is used for global interrupt enable, and to pace back-to-back interrupts using an interrupt retrigger count based on the peripheral clock (PLL1/6). There is also an 8K block of RAM local to the EMAC that is used to hold packet buffer descriptors.

Note that although the EMAC control module and the EMAC module have slightly different functions, in practice, the type of maintenance performed on the EMAC control module is more commonly conducted from the EMAC module software (as opposed to the MDIO module).

The initialization of the EMAC control module consists of two parts:

1. Configuration of the interrupt to the CPU.
2. Initialization of the EMAC control module:
   - Setting the interrupt pace count using the EMAC control module interrupt timer count register (EWINTTCNT).
   - Initializing the EMAC and MDIO modules.
   - Enabling interrupts in the EMAC control module using the EMAC control module interrupt control register (EWCTL).

When using the register-level CSL, the code to perform the actions associated with the second part may appear as in Example 4.

The process of mapping the EMAC interrupts to one of the CPU's interrupts is done using the DSP interrupt controller. Once the interrupt is mapped to a CPU interrupt, general masking and unmasking of the interrupt (to control reentrancy) should be done at the chip level by manipulating the interrupt enable mask. The EMAC control module interrupt control register (EWCTL) should only be used to enable and disable interrupts from within the EMAC interrupt service routine (ISR). This is because disabling and reenabling the interrupt in EWCTL also resets the interrupt pace counter.

Example 4. EMAC Control Module Initialization Code

```
 Uint32 tmpval ;
/*
// Globally disable EMAC/MDIO interrupts in the control module
*/
CSL_FINST( ECTL_REGS->EWCTL, ECTL_EWCTL_INTEN, DISABLE ) ;

/* Wait about 100 cycles */
for( I=0; i<5; I++ )
    tmpval = ECTL_REGS->EWCTL ;
/* Set Interrupt Timer Count (PLL1clk/6) */
ECTL_REGS->EWINTTCNT = 1500 ;
/*
// Initialize MDIO and EMAC Module
*/[Discussed later in this document]
/* Enable global interrupt in the control module */
CSL_FINST( ECTL_REGS->EWCTL, ECTL_EWCTL_INTEN, ENABLE ) ;
```

2.15.3 MDIO Module Initialization

The MDIO module is used to initially configure and monitor one or more external PHY devices. Other than initializing the software state machine (details on this state machine can be found in the IEEE 802.3 standard), all that needs to be done for the MDIO module is to enable the MDIO engine and to configure the clock divider. To set the clock divider, supply an MDIO clock of 1 MHZ. For example, since the base clock used is the peripheral clock (PLL1/6), for a processor operating at a PLL frequency of 594 MHZ the divider can be set to 99, with slower MDIO clocks for slower peripheral clock frequencies being perfectly acceptable.

Both the state machine enable and the MDIO clock divider are controlled through the MDIO control register (CONTROL). If none of the potentially connected PHYs require the access preamble, the PREAMBLE bit in CONTROL can also be set to speed up PHY register access. The code for this may appear as in Example 5.

Example 5. MDIO Module Initialization Code

```
  #define PCLK 99
...
/* Enable MDIO and setup divider */
MDIO_REGS->CONTROL =    CSL_FMKT( MDIO_CONTROL_ENABLE, YES) |
                              CSL_FMK( MDIO_CONTROL_CLKDIV, PCLK ) ;
```

If the MDIO module is to operate on an interrupt basis, the interrupts can be enabled at this time using the MDIO user command complete interrupt mask set register (USERINTMASKSET) for register access and the MDIO user PHY select register (USERPHYSEL*n*) if a target PHY is already known.

Once the MDIO state machine has been initialized and enabled, it starts polling all 32 PHY addresses on the MDIO bus, looking for an active PHY. Since it can take up to 50 µs to read one register, it can be some time before the MDIO module provides an accurate representation of whether a PHY is available. Also, a PHY can take up to 3 seconds to negotiate a link. Thus, it is advisable to run the MDIO software off a time-based event rather than polling.

For more information on PHY control registers, see your PHY device documentation.

2.15.4 EMAC Module Initialization

The EMAC module is used to send and receive data packets over the network. This is done by maintaining up to eight transmit and receive descriptor queues. The EMAC module configuration must also be kept up-to-date based on PHY negotiation results returned from the MDIO module. Most of the work in developing an application or device driver for Ethernet is programming this module.

The following is the initialization procedure a device driver would follow to get the EMAC to the state where it is ready to receive and send Ethernet packets. Some of these steps are not necessary when performed immediately after device reset.

1. If enabled, clear the device interrupt enable in the EMAC control module interrupt control register (EWCTL).
2. Clear the MAC control register (MACCONTROL), receive control register (RXCONTROL), and transmit control register (TXCONTROL) (not necessary immediately after reset).
3. Initialize all 16 header descriptor pointer registers (RX*n*HDP and TX*n*HDP) to 0.
4. Clear all 36 statistics registers by writing 0 (not necessary immediately after reset).
5. Setup the local Ethernet MAC address by programming the MAC index register (MACINDEX), MAC address high bytes register (MACADDRHI), and MAC address low bytes register (MACADDRLO). Be sure to program all eight MAC addresses - whether the receive channel is to be enabled or not. Duplicate the same MAC address across all unused channels. When using more than one receive channel, start with channel 0 and progress upwards.
6. Initialize the receive channel *n* free buffer count registers (RX*n*FREEBUFFER), receive channel *n* flow control threshold register (RX*n*FLOWTHRESH), and receive filter low priority frame threshold register (RXFILTERLOWTHRESH), if buffer flow control is to be enabled.
7. Most device drivers open with no multicast addresses, so clear the MAC address hash registers (MACHASH1 and MACHASH2) to 0.
8. Write the receive buffer offset register (RXBUFFEROFFSET) value (typically zero).
9. Initially clear all unicast channels by writing FFh to the receive unicast clear register (RXUNICASTCLEAR). If unicast is desired, it can be enabled now by writing the receive unicast set register (RXUNICASTSET). Some drivers will default to unicast on device open while others will not.
10. Setup the receive multicast/broadcast/promiscuous channel enable register (RXMBPENABLE) with an initial configuration. The configuration is based on the current receive filter settings of the device driver. Some drivers may enable things like broadcast and multicast packets immediately, while others may not.
11. Set the appropriate configuration bits in MACCONTROL (do not set the GMIIEN bit yet).
12. Clear all unused channel interrupt bits by writing the receive interrupt mask clear register (RXINTMASKCLEAR) and the transmit interrupt mask clear register (TXINTMASKCLEAR).
13. Enable the receive and transmit channel interrupt bits in the receive interrupt mask set register (RXINTMASKSET) and the transmit interrupt mask set register (TXINTMASKSET) for the channels to be used, and enable the HOSTMASK and STATMASK bits using the MAC interrupt mask set register (MACINTMASKSET).
14. Initialize the receive and transmit descriptor list queues.
15. Prepare receive by writing a pointer to the head of the receive buffer descriptor list to RX*n*HDP.
16. Enable the receive and transmit DMA controllers by setting the RXEN bit in RXCONTROL and the TXEN bit in TXCONTROL. Then set the GMIIEN bit in MACCONTROL.
17. Enable the device interrupt in EWCTL.

2.16 Interrupt Support

2.16.1 EMAC Module Interrupt Events and Requests

The EMAC module generates 18 interrupt events:

- TXPEND*n*: Transmit packet completion interrupt for transmit channels 0 through 7
- RXPEND*n*: Receive packet completion interrupt for receive channels 0 through 7
- STATPEND: Statistics interrupt
- HOSTPEND: Host error interrupt

2.16.1.1 Transmit Packet Completion Interrupts

The transmit DMA engine has eight channels, with each channel having a corresponding interrupt (TXPEND*n*). The transmit interrupts are level interrupts that remain asserted until cleared by the CPU.

Each of the eight transmit channel interrupts may be individually enabled by setting the appropriate bit in the transmit interrupt mask set register (TXINTMASKSET) to 1. Each of the eight transmit channel interrupts may be individually disabled by clearing the appropriate bit in the transmit interrupt mask clear register (TXINTMASKCLEAR) to 0. The raw and masked transmit interrupt status may be read by reading the transmit interrupt status (unmasked) register (TXINTSTATRAW) and the transmit interrupt status (masked) register (TXINTSTATMASKED), respectively.

When the EMAC completes the transmission of a packet, the EMAC issues an interrupt to the CPU by writing the packet's last buffer descriptor address to the appropriate channel queue's transmit completion pointer located in the state RAM block. The interrupt is generated by the write when enabled by the interrupt mask, regardless of the value written.

Upon interrupt reception, the CPU processes one or more packets from the buffer chain and then acknowledges an interrupt by writing the address of the last buffer descriptor processed to the queue's associated transmit completion pointer in the transmit DMA state RAM.

The data written by the host (buffer descriptor address of the last processed buffer) is compared to the data in the register written by the EMAC port (address of last buffer descriptor used by the EMAC). If the two values are not equal (which means that the EMAC has transmitted more packets than the CPU has processed interrupts for), the transmit packet completion interrupt signal remains asserted. If the two values are equal (which means that the host has processed all packets that the EMAC has transferred), the pending interrupt is cleared. The value that the EMAC is expecting is found by reading the transmit channel *n* completion pointer register (TX*n*CP).

The EMAC write to the completion pointer actually stores the value in the state RAM. The CPU written value does not actually change the register value. The host written value is compared to the register content (which was written by the EMAC) and if the two values are equal then the interrupt is removed; otherwise, the interrupt remains asserted. The host may process multiple packets prior to acknowledging an interrupt, or the host may acknowledge interrupts for every packet.

2.16.1.2 Receive Packet Completion Interrupts

The receive DMA engine has eight channels, which each channel having a corresponding interrupt (RXPEND*n*). The receive interrupts are level interrupts that remain asserted until cleared by the CPU.

Each of the eight receive channel interrupts may be individually enabled by setting the appropriate bit in the receive interrupt mask set register (RXINTMASKSET) to 1. Each of the eight receive channel interrupts may be individually disabled by clearing the appropriate bit in the receive interrupt mask clear register (RXINTMASKCLEAR) to 0. The raw and masked receive interrupt status may be read by reading the receive interrupt status (unmasked) register (RXINTSTATRAW) and the receive interrupt status (masked) register (RXINTSTATMASKED), respectively.

When the EMAC completes a packet reception, the EMAC issues an interrupt to the CPU by writing the packet's last buffer descriptor address to the appropriate channel queue's receive completion pointer located in the state RAM block. The interrupt is generated by the write when enabled by the interrupt mask, regardless of the value written.

Upon interrupt reception, the CPU processes one or more packets from the buffer chain and then acknowledges one or more interrupt(s) by writing the address of the last buffer descriptor processed to the queue's associated receive completion pointer in the receive DMA state RAM.

The data written by the host (buffer descriptor address of the last processed buffer) is compared to the data in the register written by the EMAC (address of last buffer descriptor used by the EMAC). If the two values are not equal (which means that the EMAC has received more packets than the CPU has processed interrupts for), the receive packet completion interrupt signal remains asserted. If the two values are equal (which means that the host has processed all packets that the EMAC has received), the pending interrupt is de-asserted. The value that the EMAC is expecting is found by reading the receive channel *n* completion pointer register (RX*n*CP).

The EMAC write to the completion pointer actually stores the value in the state RAM. The CPU written value does not actually change the register value. The host written value is compared to the register content (which was written by the EMAC) and if the two values are equal then the interrupt is removed; otherwise, the interrupt remains asserted. The host may process multiple packets prior to acknowledging an interrupt, or the host may acknowledge interrupts for every packet.

2.16.1.3 Statistics Interrupt

The statistics level interrupt (STATPEND) is issued when any statistics value is greater than or equal to 8000 0000h, if enabled by setting the STATMASK bit in the MAC interrupt mask set register (MACINTMASKSET) to 1. The statistics interrupt is removed by writing to decrement any statistics value greater than 8000 0000h. As long as the most-significant bit of any statistics value is set, the interrupt remains asserted.

2.16.1.4 Host Error Interrupt

The host error interrupt (HOSTPEND) is issued, if enabled, under error conditions dealing with the handling of buffer descriptors, detected during transmit or receive DMA transactions. The failure of the software application to supply properly formatted buffer descriptors results in this error. The error bit can only be cleared by resetting the EMAC module in hardware.

The host error interrupt is enabled by setting the HOSTMASK bit in the MAC interrupt mask set register (MACINTMASKSET) to 1. The host error interrupt is disabled by clearing the appropriate bit in the MAC interrupt mask clear register (MACINTMASKCLEAR) to 0. The raw and masked host error interrupt status may be read by reading the MAC interrupt status (unmasked) register (MACINTSTATRAW) and the MAC interrupt status (masked) register (MACINTSTATMASKED), respectively.

The transmit host error conditions are:

- SOP error
- Ownership bit not set in SOP buffer
- Zero next buffer descriptor pointer with EOP
- Zero buffer pointer
- Zero buffer length
- Packet length error

The receive host error conditions are:

- Ownership bit not set in input buffer
- Zero buffer pointer

2.16.2 MDIO Module Interrupt Events and Requests

The MDIO module generates two interrupt events:

- LINKINT: Serial interface link change interrupt. Indicates a change in the state of the PHY link
- USERINT: Serial interface user command event complete interrupt

2.16.2.1 Link Change Interrupt

The MDIO module asserts a link change interrupt (LINKINT) if there is a change in the link state of the PHY corresponding to the address in the PHYADRMON bit in the MDIO user PHY select register *n* (USERPHYSEL*n*), and if the LINKINTENB bit is also set in USERPHYSEL*n*. This interrupt event is also captured in the LINKINTRAW bit in the MDIO link status change interrupt register (LINKINTRAW). LINKINTRAW bits 0 and 1 correspond to USERPHYSEL0 and USERPHYSEL1, respectively.

When the interrupt is enabled and generated, the corresponding LINKINTMASKED bit is also set in the MDIO link status change interrupt register (LINKINTMASKED). The interrupt is cleared by writing back the same bit to LINKINTMASKED (write to clear).

2.16.2.2 User Access Completion Interrupt

When the GO bit in one of the MDIO user access registers (USERACCESS*n*) transitions from 1 to 0 (indicating completion of a user access) and the corresponding USERINTMASKSET bit in the MDIO user command complete interrupt mask set register (USERINTMASKSET) corresponding to USERACCESS0 or USERACCESS1 is set, a user access completion interrupt (USERINT) is asserted. This interrupt event is also captured in the USERINTRAW bit in the MDIO user command complete interrupt register (USERINTRAW). USERINTRAW bits 0 and bit 1 correspond to USERACCESS0 and USERACCESS1, respectively.

When the interrupt is enabled and generated, the corresponding USERINTMASKED bit is also set in the MDIO user command complete interrupt register (USERINTMASKED). The interrupt is cleared by writing back the same bit to USERINTMASKED (write to clear).

2.16.3 Proper Interrupt Processing

All the interrupts signaled from the EMAC and MDIO modules are level driven, so if they remain active, their level remains constant; the CPU core requires edge-triggered interrupts. In order to properly convert the level-driven interrupt signal to an edge-triggered signal, the application software must make use of the interrupt control logic contained in the EMAC control module.

Section 2.6.3 discusses the interrupt control contained in the EMAC control module. For safe interrupt processing, upon entry to the ISR, the software application should disable interrupts using the EMAC control module interrupt control register (EWCTL), and then reenable them upon leaving the ISR. If any interrupt signals are active at that time, this creates another rising edge on the interrupt signal going to the CPU interrupt controller, thus triggering another interrupt. The EMAC control module also uses the EMAC control module interrupt timer count register (EWINTTCNT) to implement interrupt pacing.

2.16.4 Interrupt Multiplexing

The EMAC control module combines different interrupt signals from both the EMAC and MDIO modules and generates a single interrupt signal that is wired to the CPU interrupt controller. Once this interrupt is generated, the reason for the interrupt can be read from the MAC input vector register (MACINVECTOR) located in the EMAC memory map. MACINVECTOR combines the status of the following 20 interrupt signals: TXPEND*n*, RXPEND*n*, STATPEND, HOSTPEND, LINKINT, and USERINT.

The EMAC and MDIO interrupts are combined within the EMAC control module and mapped to the DSP interrupt INT43 through the DSP interrupt controller. For more details on the DSP interrupt controller, see the *TMS320C64x+ DSP Megamodule Reference Guide* (SPRU871).

2.17 Power Management

Each of the three main components of the EMAC peripheral can independently be placed in reduced-power modes to conserve power during periods of low activity. The power management of the EMAC peripheral is controlled by the processor Power and Sleep Controller (PSC). The PSC acts as a master controller for power management on behalf of all of the peripherals on the device.

The power conservation modes available for each of the three components of the EMAC/MDIO peripheral are:

- *Idle/Disabled state*. This mode stops the clocks going to the peripheral, and prevents all the register accesses. After reenabling the peripheral from this idle state, all the registers values prior to setting into the disabled state are restored, and data transmission can proceed. No reinitialization is required.
- *Synchronized reset.* This state is similar to the Power-on Reset (POR) state, when the processor is turned-on; reset to the peripheral is asserted, and clocks to the peripheral are gated after that. The registers are reset to their default value. When powering-up after a synchronized reset, all the EMAC submodules need to be reinitialized before any data transmission can happen.

For more information on the use of the processor Power and Sleep Controller (PSC), see the *TMS320DM643x DMP DSP Subsystem Reference Guide* (SPRU978).

2.18 Emulation Considerations

EMAC emulation control is implemented for compatibility with other peripherals. The SOFT and FREE bits in the emulation control register (EMCONTROL) allow EMAC operation to be suspended.

When the emulation suspend state is entered, the EMAC stops processing receive and transmit frames at the next frame boundary. Any frame currently in reception or transmission is completed normally without suspension. For transmission, any complete or partial frame in the transmit cell FIFO is transmitted. For receive, frames that are detected by the EMAC after the suspend state is entered are ignored. No statistics are kept for ignored frames.

Table 6 shows how the SOFT and FREE bits affect the operation of the emulation suspend.

Note: Emulation suspend has not been tested.

Table 6. Emulation Control

SOFT	FREE	Description
0	0	Normal operation
1	0	Emulation suspend
X	1	Normal operation

3 EMAC Control Module Registers

Table 7 lists the memory-mapped registers for the EMAC control module. See the device-specific data manual for the memory address of these registers.

Table 7. EMAC Control Module Registers

Offset	Acronym	Register Description	Section
04h	EWCTL	EMAC Control Module Interrupt Control Register	Section 3.1
08h	EWINTTCNT	EMAC Control Module Interrupt Timer Count Register	Section 3.2

3.1 EMAC Control Module Interrupt Control Register (EWCTL)

The EMAC control module interrupt control register (EWCTL) is used to enable and disable the central interrupt from the EMAC and MDIO modules.

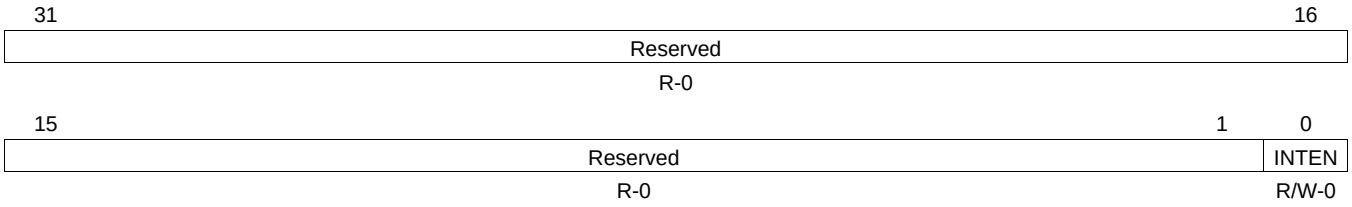
It is expected that any time the EMAC and MDIO interrupt is being serviced, the software disables the INTEN bit in EWCTL. This ensures that the interrupt line goes back to zero. The software re-enables the INTEN bit after clearing all the pending interrupts and before leaving the interrupt service routine. At this point, if the EMAC control module monitors any interrupts still pending, it reasserts the interrupt line, and generates a new edge that the CPU can recognize.

Any time the INTEN bit is cleared to 0, the EMAC_MDIO_INT signal to the CPU is kept deasserted. If the INTEN bit is set to 1, then the interrupt control logic checks all the interrupt lines from EMAC and MDIO. If any of these interrupt lines are active, the EMAC_MDIO_INT signal is asserted. Assertion of this signal generates an edge, which can then be recognized as a valid interrupt by the CPU.

The INTEN bit takes care of two problems associated with level interrupts from the EMAC and the MDIO modules. First, it makes sure that none of the interrupts are missed; second, it makes sure that only the required number of interrupts are sent to the CPU.

The EWCTL is shown in Figure 11 and described in Table 8.

Figure 11. EMAC Control Module Interrupt Control Register (EWCTL)



LEGEND: R = Read only; R/W = Read/Write; -*n* = value after reset

Table 8. EMAC Control Module Interrupt Control Register (EWCTL) Field Descriptions

Bit	Field	Value	Description
31-1	Reserved	0	Reserved
0	INTEN		Controls the EMAC_MDIO_INT interrupt generation to the CPU.
		0	EMAC and MDIO interrupts are disabled.
		1	EMAC and MDIO interrupts are enabled.

3.2 EMAC Control Module Interrupt Timer Count Register (EWINTTCNT)

The EMAC control module interrupt timer count register (EWINTTCNT) is used to control the generation of back-to-back interrupts from the EMAC and MDIO modules. The value of this timer count is loaded into an internal counter every time interrupts are enabled using the INTEN bit in the EMAC control module interrupt control register (EWCTL). A second interrupt cannot be generated until this count reaches 0. The counter is decremented at a frequency of PLL1clock/6; the default reset count is 0 (inactive) and the maximum value is 1 FFFFh (131 071).

The EWINTTCNT is shown in Figure 12 and described in Table 9.

Figure 12. EMAC Control Module Interrupt Timer Count Register (EWINTTCNT)

31	17	16
Reserved		EWINTTCNT
R-0		R/W-0

15	0
EWINTTCNT	
R/W-0	

LEGEND: R = Read only; R/W = Read/Write; -*n* = value after reset

Table 9. EMAC Control Module Interrupt Timer Count Register (EWINTTCNT) Field Descriptions

Bit	Field	Value	Description
31-18	Reserved	0	Reserved
17-0	EWINTTCNT	0-1 FFFFh	Interrupt timer count. EWINTTCNT is a 17-bit interrupt timer count that is used to control the generation of back-to-back interrupts from the EMAC and MDIO modules. The value of EWINTTCNT is loaded in an internal time counter every time interrupts are enabled by writing a 1 to the INTEN bit in EWCTL (note the INTEN bit must transition from 0 to 1 to initialize the internal time counter). Once initialized, the time counter will count down with each peripheral clock until it reaches 0. A second interrupt cannot be generated until this counter reaches 0. Any time the time counter has a non-zero value, the interrupt logic will block the EMAC_MDIO_INT interrupt to the CPU. Thus, if any of the interrupts coming to the EMAC control module is asserted, the interrupt logic will assert the EMAC_MDIO_INT signal to the CPU, provided the INTEN bit in EWCTL is set, and the time counter value is 0.

4 MDIO Registers

Table 10 lists the memory-mapped registers for the MDIO module. See the device-specific data manual for the memory address of these registers.

Table 10. Management Data Input/Output (MDIO) Registers

Offset	Acronym	Register Description	Section
0h	VERSION	MDIO Version Register	Section 4.1
4h	CONTROL	MDIO Control Register	Section 4.2
8h	ALIVE	PHY Alive Status register	Section 4.3
Ch	LINK	PHY Link Status Register	Section 4.4
10h	LINKINTRAW	MDIO Link Status Change Interrupt (Unmasked) Register	Section 4.5
14h	LINKINTMASKED	MDIO Link Status Change Interrupt (Masked) Register	Section 4.6
20h	USERINTRAW	MDIO User Command Complete Interrupt (Unmasked) Register	Section 4.7
24h	USERINTMASKED	MDIO User Command Complete Interrupt (Masked) Register	Section 4.8
28h	USERINTMASKSET	MDIO User Command Complete Interrupt Mask Set Register	Section 4.9
2Ch	USERINTMASKCLEAR	MDIO User Command Complete Interrupt Mask Clear Register	Section 4.10
80h	USERACCESS0	MDIO User Access Register 0	Section 4.11
84h	USERPHYSEL0	MDIO User PHY Select Register 0	Section 4.12
88h	USERACCESS1	MDIO User Access Register 1	Section 4.13
8Ch	USERPHYSEL1	MDIO User PHY Select Register 1	Section 4.14

4.1 MDIO Version Register (VERSION)

The MDIO version register (VERSION) is shown in Figure 13 and described in Table 11.

Figure 13. MDIO Version Register (VERSION)

31	16
MODID	
R-7h	

15	8	7	0
REVMAJ		REVMIN	
R-1h		R-3h	

LEGEND: R = Read only; -*n* = value after reset

Table 11. MDIO Version Register (VERSION) Field Descriptions

Bit	Field	Value	Description
31-16	MODID		Identifies type of peripheral.
		7h	MDIO
15-8	REVMAJ		Identifies major revision of peripheral. Revisions are indicated by a revision code taking the format REVMAJ.REVMIN.
		1h	Current major revision of peripheral.
7-0	REVMIN		Identifies minor revision of peripheral. Revisions are indicated by a revision code taking the format REVMAJ.REVMIN.
		3h	Current minor revision of peripheral.

4.2 MDIO Control Register (CONTROL)

The MDIO control register (CONTROL) is shown in Figure 14 and described in Table 12.

Figure 14. MDIO Control Register (CONTROL)

31	30	29	28 24	23 21	20	19	18	17 16
IDLE	ENABLE	Rsvd	HIGHEST_USER_CHANNEL	Reserved	PREAMBLE	FAULT	FAULTENB	Reserved
R-1	R/W-0	R-0	R-1	R-0	R/W-0	R/WC-0	R/W-0	R-0

15 0
CLKDIV
R/W-FFh

LEGEND: R/W = R = Read only; R/W = Read/Write; WC = Write 1 to clear; *-n* = value after reset

Table 12. MDIO Control Register (CONTROL) Field Descriptions

Bit	Field	Value	Description
31	IDLE		State machine IDLE status bit.
		0	State machine is not in idle state.
		1	State machine is in idle state.
30	ENABLE		State machine enable control bit. If the MDIO state machine is active at the time it is disabled, it will complete the current operation before halting and setting the idle bit.
		0	Disables the MDIO state machine.
		1	Enable the MDIO state machine.
29	Reserved	0	Reserved
28-24	HIGHEST_USER_CHANNEL	0-1Fh	Highest user channel that is available in the module. It is currently set to 1. This implies that MDIOUserAccess1 is the highest available user access channel.
23-21	Reserved	0	Reserved
20	PREAMBLE		Preamble disable
		0	Standard MDIO preamble is used.
		1	Disables this device from sending MDIO frame preambles.
19	FAULT		Fault indicator. This bit is set to 1 if the MDIO pins fail to read back what the device is driving onto them. This indicates a physical layer fault and the module state machine is reset. Writing a 1 to it clears this bit.
		0	No failure
		1	Physical layer fault; the MDIO state machine is reset.
18	FAULTENB		Fault detect enable. This bit has to be set to 1 to enable the physical layer fault detection.
		0	Disables the physical layer fault detection.
		1	Enables the physical layer fault detection.
17-16	Reserved	0	Reserved
15-0	CLKDIV	0-FFFFh	Clock Divider bits. This field specifies the division ratio between the peripheral clock and the frequency of MDCLK. MDCLK is disabled when CLKDIV is cleared to 0. MDCLK frequency = peripheral clock frequency/(CLKDIV + 1).

4.3 PHY Acknowledge Status Register (ALIVE)

The PHY acknowledge status register (ALIVE) is shown in Figure 15 and described in Table 13.

Figure 15. PHY Acknowledge Status Register (ALIVE)

31	16
ALIVE	
R/WC-0	

15	0
ALIVE	
R/WC-0	

LEGEND: R/W = Read/Write; WC = Write 1 to clear; -*n* = value after reset

Table 13. PHY Acknowledge Status Register (ALIVE) Field Descriptions

Bit	Field	Value	Description
31-0	ALIVE		MDIO Alive bits. Each of the 32 bits of this register is set if the most recent access to the PHY with address corresponding to the register bit number was acknowledged by the PHY; the bit is reset if the PHY fails to acknowledge the access. Both the user and polling accesses to a PHY will cause the corresponding alive bit to be updated. The alive bits are only meant to be used to give an indication of the presence or not of a PHY with the corresponding address. Writing a 1 to any bit will clear it, writing a 0 has no effect.
		0	The PHY fails to acknowledge the access.
		1	The most recent access to the PHY with an address corresponding to the register bit number was acknowledged by the PHY.

4.4 PHY Link Status Register (LINK)

The PHY link status register (LINK) is shown in Figure 16 and described in Table 14.

Figure 16. PHY Link Status Register (LINK)

31	16
LINK	
R-0	

15	0
LINK	
R-0	

LEGEND: R = Read only; -*n* = value after reset

Table 14. PHY Link Status Register (LINK) Field Descriptions

Bit	Field	Value	Description
31-0	LINK		MDIO Link state bits. This register is updated after a read of the generic status register of a PHY. The bit is set if the PHY with the corresponding address has link and the PHY acknowledges the read transaction. The bit is reset if the PHY indicates it does not have link or fails to acknowledge the read transaction. Writes to the register have no effect.
		0	The PHY indicates it does not have a link or fails to acknowledge the read transaction
		1	The PHY with the corresponding address has a link and the PHY acknowledges the read transaction.

4.5 MDIO Link Status Change Interrupt (Unmasked) Register (LINKINTRAW)

The MDIO link status change interrupt (unmasked) register (LINKINTRAW) is shown in Figure 17 and described in Table 15.

Figure 17. MDIO Link Status Change Interrupt (Unmasked) Register (LINKINTRAW)

31	16
Reserved	
R-0	

15	2	1	0
Reserved		LINKINTRAW	
R-0		R/WC-0	

LEGEND: R = Read only; R/W = Read/Write; WC = Write 1 to clear; -*n* = value after reset

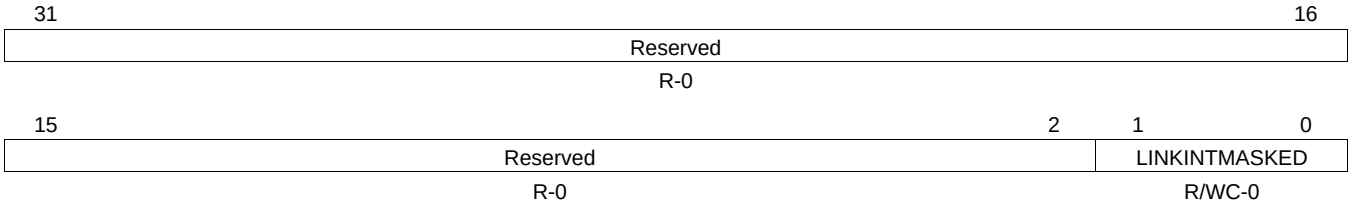
Table 15. MDIO Link Status Change Interrupt (Unmasked) Register (LINKINTRAW) Field Descriptions

Bit	Field	Value	Description
31-2	Reserved	0	Reserved
1-0	LINKINTRAW		MDIO Link change event, raw value. When asserted, a bit indicates that there was an MDIO link change event (that is, change in the LINK register) corresponding to the PHY address in USERPHYSEL. LINKINTRAW[0] and LINKINTRAW[1] correspond to USERPHYSEL0 and USERPHYSEL1, respectively. Writing a 1 will clear the event and writing a 0 has no effect.
		0	No MDIO link change event.
		1	An MDIO link change event (change in the LINK register) corresponding to the PHY address in MDIO user PHY select register *n* (USERPHYSEL*n*).

4.6 MDIO Link Status Change Interrupt (Masked) Register (LINKINTMASKED)

The MDIO link status change interrupt (masked) register (LINKINTMASKED) is shown in Figure 18 and described in Table 16.

Figure 18. MDIO Link Status Change Interrupt (Masked) Register (LINKINTMASKED)



LEGEND: R = Read only; R/W = Read/Write; WC = Write 1 to clear; -*n* = value after reset

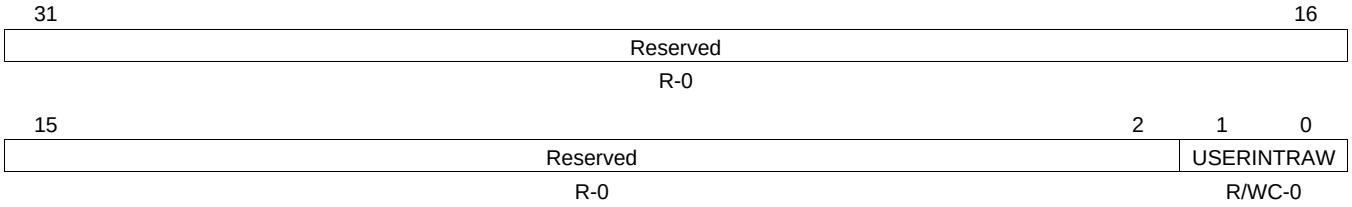
Table 16. MDIO Link Status Change Interrupt (Masked) Register (LINKINTMASKED) Field Descriptions

Bit	Field	Value	Description
31-2	Reserved	0	Reserved
1-0	LINKINTMASKED		MDIO Link change interrupt, masked value. When asserted, a bit indicates that there was an MDIO link change event (that is, change in the LINK register) corresponding to the PHY address in USERPHYSEL and the corresponding LINKINTENB bit was set. LINKINTMASKED[0] and LINKINTMASKED[1] correspond to USERPHYSEL0 and USERPHYSEL1, respectively. Writing a 1 will clear the event and writing a 0 has no effect.
		0	No MDIO link change event.
		1	An MDIO link change event (change in the LINK register) corresponding to the PHY address in MDIO user PHY select register *n* (USERPHYSEL*n*) and the LINKINTENB bit in USERPHYSEL*n* is set to 1.

4.7 MDIO User Command Complete Interrupt (Unmasked) Register (USERINTRAW)

The MDIO user command complete interrupt (unmasked) register (USERINTRAW) is shown in Figure 19 and described in Table 17.

Figure 19. MDIO User Command Complete Interrupt (Unmasked) Register (USERINTRAW)



LEGEND: R = Read only; R/W = Read/Write; WC = Write 1 to clear; -*n* = value after reset

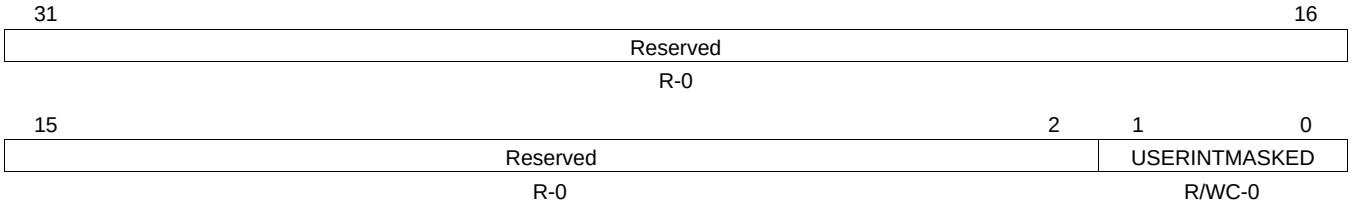
Table 17. MDIO User Command Complete Interrupt (Unmasked) Register (USERINTRAW) Field Descriptions

Bit	Field	Value	Description
31-2	Reserved	0	Reserved
1-0	USERINTRAW		MDIO User command complete event bits. When asserted, a bit indicates that the previously scheduled PHY read or write command using that particular USERACCESS register has completed. USERINTRAW[0] and USERINTRAW[1] correspond to USERACCESS0 and USERACCESS1, respectively. Writing a 1 will clear the event and writing a 0 has no effect.
		0	No MDIO user command complete event.
		1	The previously scheduled PHY read or write command using MDIO user access register *n* (USERACCESS*n*) has completed.

4.8 MDIO User Command Complete Interrupt (Masked) Register (USERINTMASKED)

The MDIO user command complete interrupt (masked) register (USERINTMASKED) is shown in Figure 20 and described in Table 18.

Figure 20. MDIO User Command Complete Interrupt (Masked) Register (USERINTMASKED)



LEGEND: R = Read only; R/W = Read/Write; WC = Write 1 to clear; -*n* = value after reset

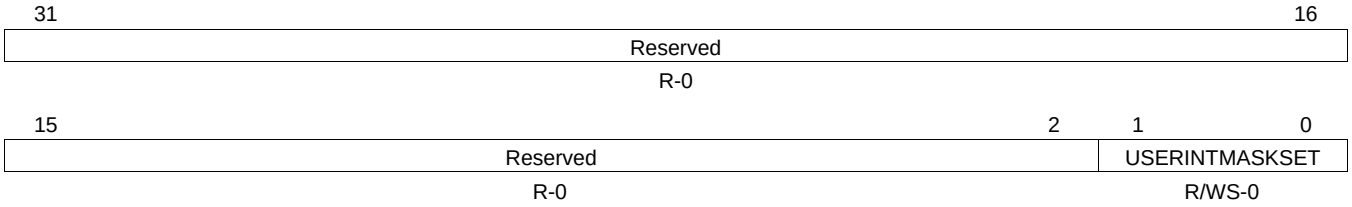
Table 18. MDIO User Command Complete Interrupt (Masked) Register (USERINTMASKED) Field Descriptions

Bit	Field	Value	Description
31-2	Reserved	0	Reserved
1-0	USERINTMASKED		Masked value of MDIO User command complete interrupt. When asserted, a bit indicates that the previously scheduled PHY read or write command using that particular USERACCESS register has completed and the corresponding USERINTMASKSET bit is set to 1. USERINTMASKED[0] and USERINTMASKED[1] correspond to USERACCESS0 and USERACCESS1, respectively. Writing a 1 will clear the interrupt and writing a 0 has no effect.
		0	No MDIO user command complete event.
		1	The previously scheduled PHY read or write command using MDIO user access register *n* (USERACCESS*n*) has completed and the corresponding bit in USERINTMASKSET is set to 1.

4.9 MDIO User Command Complete Interrupt Mask Set Register (USERINTMASKSET)

The MDIO user command complete interrupt mask set register (USERINTMASKSET) is shown in Figure 21 and described in Table 19.

Figure 21. MDIO User Command Complete Interrupt Mask Set Register (USERINTMASKSET)



LEGEND: R = Read only; R/W = Read/Write; WS = Write 1 to set; -*n* = value after reset

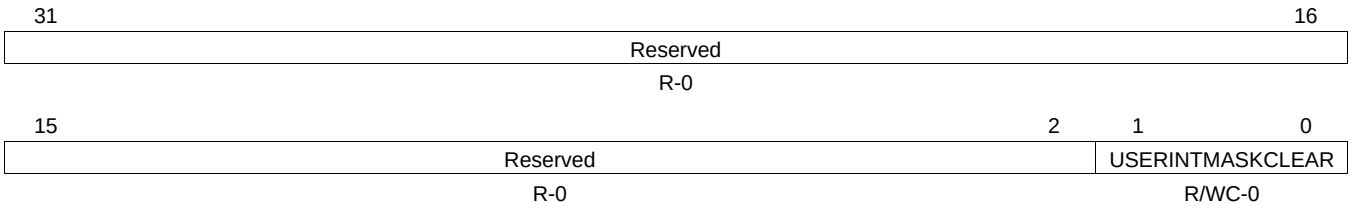
Table 19. MDIO User Command Complete Interrupt Mask Set Register (USERINTMASKSET) Field Descriptions

Bit	Field	Value	Description
31-2	Reserved	0	Reserved
1-0	USERINTMASKSET		MDIO user interrupt mask set for USERINTMASKED[1:0], respectively. Setting a bit to 1 will enable MDIO user command complete interrupts for that particular USERACCESS register. MDIO user interrupt for a particular USERACCESS register is disabled if the corresponding bit is 0. USERINTMASKSET[0] and USERINTMASKSET[1] correspond to USERACCESS0 and USERACCESS1, respectively. Writing a 0 to this register has no effect.
		0	MDIO user command complete interrupts for the MDIO user access register *n* (USERACCESS*n*) are disabled.
		1	MDIO user command complete interrupts for the MDIO user access register *n* (USERACCESS*n*) are enabled.

4.10 MDIO User Command Complete Interrupt Mask Clear Register (USERINTMASKCLEAR)

The MDIO user command complete interrupt mask clear register (USERINTMASKCLEAR) is shown in Figure 22 and described in Table 20.

Figure 22. MDIO User Command Complete Interrupt Mask Clear Register (USERINTMASKCLEAR)



LEGEND: R = Read only; R/W = Read/Write; WC = Write 1 to clear; -*n* = value after reset

Table 20. MDIO User Command Complete Interrupt Mask Clear Register (USERINTMASKCLEAR) Field Descriptions

Bit	Field	Value	Description
31-2	Reserved	0	Reserved
1-0	USERINTMASKCLEAR		MDIO user command complete interrupt mask clear for USERINTMASKED[1:0], respectively. Setting a bit to 1 will disable further user command complete interrupts for that particular USERACCESS register. USERINTMASKCLEAR[0] and USERINTMASKCLEAR[1] correspond to USERACCESS0 and USERACCESS1, respectively. Writing a 0 to this register has no effect.
		0	MDIO user command complete interrupts for the MDIO user access register *n* (USERACCESS*n*) are enabled.
		1	MDIO user command complete interrupts for the MDIO user access register *n* (USERACCESS*n*) are disabled.

4.11 MDIO User Access Register 0 (USERACCESS0)

The MDIO user access register 0 (USERACCESS0) is shown in Figure 23 and described in Table 21.

Figure 23. MDIO User Access Register 0 (USERACCESS0)

31	30	29	28 26	25 21	20 16
GO	WRITE	ACK	Reserved	REGADR	PHYADR
R/WS-0	R/W-0	R/W-0	R-0	R/W-0	R/W-0

15 0
DATA
R/W-0

LEGEND: R = Read only; R/W = Read/Write; WS = Write 1 to set; -*n* = value after reset

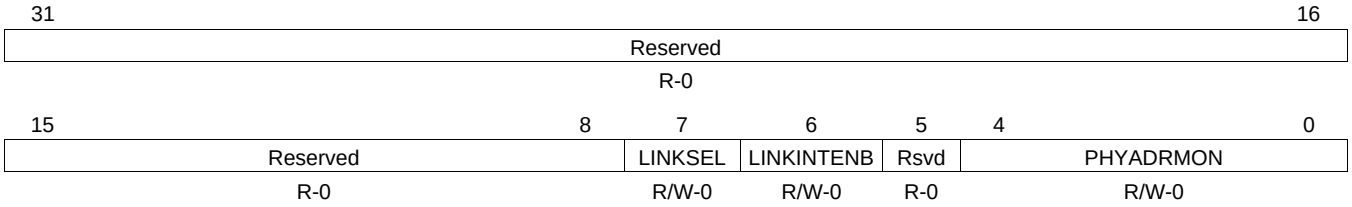
Table 21. MDIO User Access Register 0 (USERACCESS0) Field Descriptions

Bit	Field	Value	Description
31	GO	0-1	Go bit. Writing a 1 to this bit causes the MDIO state machine to perform an MDIO access when it is convenient for it to do so; this is not an instantaneous process. Writing a 0 to this bit has no effect. This bit is writeable only if the MDIO state machine is enabled. This bit will self clear when the requested access has been completed. Any writes to USERACCESS0 are blocked when the GO bit is 1.
30	WRITE		Write enable bit. Setting this bit to 1 causes the MDIO transaction to be a register write; otherwise, it is a register read.
		0	The user command is a read operation.
		1	The user command is a write operation.
29	ACK	0-1	Acknowledge bit. This bit is set if the PHY acknowledged the read transaction.
28-26	Reserved	0	Reserved
25-21	REGADR	0-1Fh	Register address bits. This field specifies the PHY register to be accessed for this transaction
20-16	PHYADR	0-1Fh	PHY address bits. This field specifies the PHY to be accessed for this transaction.
15-0	DATA	0-FFFFh	User data bits. These bits specify the data value read from or to be written to the specified PHY register.

4.12 MDIO User PHY Select Register 0 (USERPHYSEL0)

The MDIO user PHY select register 0 (USERPHYSEL0) is shown in Figure 24 and described in Table 22.

Figure 24. MDIO User PHY Select Register 0 (USERPHYSEL0)



LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 22. MDIO User PHY Select Register 0 (USERPHYSEL0) Field Descriptions

Bit	Field	Value	Description
31-8	Reserved	0	Reserved
7	LINKSEL		Link status determination select bit. Default value is 0, which implies that the link status is determined by the MDIO state machine. This is the only option supported on this device.
		0	The link status is determined by the MDIO state machine.
		1	Not supported.
6	LINKINTENB		Link change interrupt enable. Set to 1 to enable link change status interrupts for PHY address specified in PHYADRMON. Link change interrupts are disabled if this bit is cleared to 0.
		0	Link change interrupts are disabled.
		1	Link change status interrupts for PHY address specified in PHYADDRMON bits are enabled.
5	Reserved	0	Reserved
4-0	PHYADRMON	0-1Fh	PHY address whose link status is to be monitored.

4.13 MDIO User Access Register 1 (USERACCESS1)

The MDIO user access register 1 (USERACCESS1) is shown in Figure 25 and described in Table 23.

Figure 25. MDIO User Access Register 1 (USERACCESS1)

31	30	29	28 26	25 21	20 16
GO	WRITE	ACK	Reserved	REGADR	PHYADR
R/WS-0	R/W-0	R/W-0	R-0	R/W-0	R/W-0

15 0
DATA
R/W-0

LEGEND: R = Read only; R/W = Read/Write; WS = Write 1 to set; -*n* = value after reset

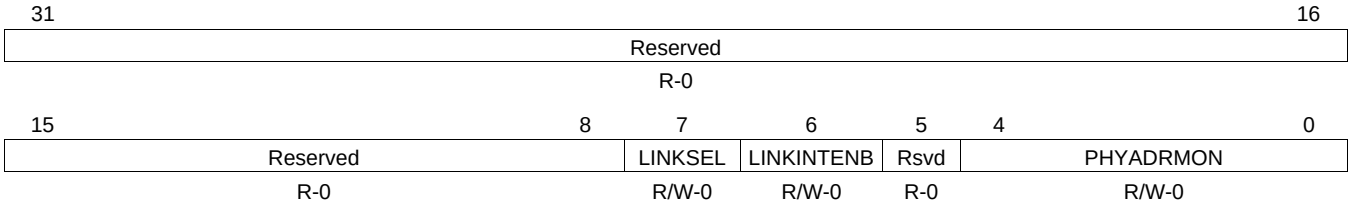
Table 23. MDIO User Access Register 1 (USERACCESS1) Field Descriptions

Bit	Field	Value	Description
31	GO	0-1	Go bit. Writing 1 to this bit causes the MDIO state machine to perform an MDIO access when it is convenient for it to do so; this is not an instantaneous process. Writing 0 to this bit has no effect. This bit is writeable only if the MDIO state machine is enabled. This bit will self clear when the requested access has been completed. Any writes to USERACCESS0 are blocked when the GO bit is 1.
30	WRITE		Write enable bit. Setting this bit to 1 causes the MDIO transaction to be a register write; otherwise, it is a register read.
		0	The user command is a read operation.
		1	The user command is a write operation.
29	ACK	0-1	Acknowledge bit. This bit is set if the PHY acknowledged the read transaction.
28-26	Reserved	0	Reserved
25-21	REGADR	0-1Fh	Register address bits. This field specifies the PHY register to be accessed for this transaction
20-16	PHYADR	0-1Fh	PHY address bits. This field specifies the PHY to be accessed for this transaction.
15-0	DATA	0-FFFFh	User data bits. These bits specify the data value read from or to be written to the specified PHY register.

4.14 MDIO User PHY Select Register 1 (USERPHYSEL1)

The MDIO user PHY select register 1 (USERPHYSEL1) is shown in Figure 26 and described in Table 24.

Figure 26. MDIO User PHY Select Register 1 (USERPHYSEL1)



LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 24. MDIO User PHY Select Register 1 (USERPHYSEL1) Field Descriptions

Bit	Field	Value	Description
31-8	Reserved	0	Reserved
7	LINKSEL		Link status determination select bit. Default value is 0, which implies that the link status is determined by the MDIO state machine. This is the only option supported on this device.
		0	The link status is determined by the MDIO state machine.
		1	Not supported.
6	LINKINTENB		Link change interrupt enable. Set to 1 to enable link change status interrupts for the PHY address specified in PHYADRMON. Link change interrupts are disabled if this bit is cleared to 0.
		0	Link change interrupts are disabled.
		1	Link change status interrupts for PHY address specified in PHYADDRMON bits are enabled.
5	Reserved	0	PHY address whose link status is to be monitored.
4-0	PHYADRMON	0-1Fh	PHY address whose link status is to be monitored.

5 Ethernet Media Access Controller (EMAC) Registers

Table 25 lists the memory-mapped registers for the EMAC. See the device-specific data manual for the memory address of these registers.

Table 25. Ethernet Media Access Controller (EMAC) Registers

Offset	Acronym	Register Description	Section
0h	TXIDVER	Transmit Identification and Version Register	Section 5.1
4h	TXCONTROL	Transmit Control Register	Section 5.2
8h	TXTEARDOWN	Transmit Teardown Register	Section 5.3
10h	RXIDVER	Receive Identification and Version Register	Section 5.4
14h	RXCONTROL	Receive Control Register	Section 5.5
18h	RXTEARDOWN	Receive Teardown Register	Section 5.6
80h	TXINTSTATRAW	Transmit Interrupt Status (Unmasked) Register	Section 5.7
84h	TXINTSTATMASKED	Transmit Interrupt Status (Masked) Register	Section 5.8
88h	TXINTMASKSET	Transmit Interrupt Mask Set Register	Section 5.9
8Ch	TXINTMASKCLEAR	Transmit Interrupt Clear Register	Section 5.10
90h	MACINVECTOR	MAC Input Vector Register	Section 5.11
A0h	RXINTSTATRAW	Receive Interrupt Status (Unmasked) Register	Section 5.12
A4h	RXINTSTATMASKED	Receive Interrupt Status (Masked) Register	Section 5.13
A8h	RXINTMASKSET	Receive Interrupt Mask Set Register	Section 5.14
ACh	RXINTMASKCLEAR	Receive Interrupt Mask Clear Register	Section 5.15
B0h	MACINTSTATRAW	MAC Interrupt Status (Unmasked) Register	Section 5.16
B4h	MACINTSTATMASKED	MAC Interrupt Status (Masked) Register	Section 5.17
B8h	MACINTMASKSET	MAC Interrupt Mask Set Register	Section 5.18
BCh	MACINTMASKCLEAR	MAC Interrupt Mask Clear Register	Section 5.19
100h	RXMBPENABLE	Receive Multicast/Broadcast/Promiscuous Channel Enable Register	Section 5.20
104h	RXUNICASTSET	Receive Unicast Enable Set Register	Section 5.21
108h	RXUNICASTCLEAR	Receive Unicast Clear Register	Section 5.22
10Ch	RXMAXLEN	Receive Maximum Length Register	Section 5.23
110h	RXBUFFEROFFSET	Receive Buffer Offset Register	Section 5.24
114h	RXFILTERLOWTHRESH	Receive Filter Low Priority Frame Threshold Register	Section 5.25
120h	RX0FLOWTHRESH	Receive Channel 0 Flow Control Threshold Register	Section 5.26
124h	RX1FLOWTHRESH	Receive Channel 1 Flow Control Threshold Register	Section 5.26
128h	RX2FLOWTHRESH	Receive Channel 2 Flow Control Threshold Register	Section 5.26
12Ch	RX3FLOWTHRESH	Receive Channel 3 Flow Control Threshold Register	Section 5.26
130h	RX4FLOWTHRESH	Receive Channel 4 Flow Control Threshold Register	Section 5.26
134h	RX5FLOWTHRESH	Receive Channel 5 Flow Control Threshold Register	Section 5.26
138h	RX6FLOWTHRESH	Receive Channel 6 Flow Control Threshold Register	Section 5.26
13Ch	RX7FLOWTHRESH	Receive Channel 7 Flow Control Threshold Register	Section 5.26
140h	RX0FREEBUFFER	Receive Channel 0 Free Buffer Count Register	Section 5.27
144h	RX1FREEBUFFER	Receive Channel 1 Free Buffer Count Register	Section 5.27
148h	RX2FREEBUFFER	Receive Channel 2 Free Buffer Count Register	Section 5.27
14Ch	RX3FREEBUFFER	Receive Channel 3 Free Buffer Count Register	Section 5.27
150h	RX4FREEBUFFER	Receive Channel 4 Free Buffer Count Register	Section 5.27
154h	RX5FREEBUFFER	Receive Channel 5 Free Buffer Count Register	Section 5.27
158h	RX6FREEBUFFER	Receive Channel 6 Free Buffer Count Register	Section 5.27
15Ch	RX7FREEBUFFER	Receive Channel 7 Free Buffer Count Register	Section 5.27
160h	MACCONTROL	MAC Control Register	Section 5.28
164h	MACSTATUS	MAC Status Register	Section 5.29

Table 25. Ethernet Media Access Controller (EMAC) Registers (continued)

Offset	Acronym	Register Description	Section
168h	EMCONTROL	Emulation Control Register	Section 5.30
16Ch	FIFOCONTROL	FIFO Control Register	Section 5.31
170h	MACCONFIG	MAC Configuration Register	Section 5.32
174h	SOFTRESET	Soft Reset Register	Section 5.33
1D0h	MACSRCADDRLO	MAC Source Address Low Bytes Register	Section 5.34
1D4h	MACSRCADDRHI	MAC Source Address High Bytes Register	Section 5.35
1D8h	MACHASH1	MAC Hash Address Register 1	Section 5.36
1DCh	MACHASH2	MAC Hash Address Register 2	Section 5.37
1E0h	BOFFTEST	Back Off Test Register	Section 5.38
1E4h	TPACETEST	Transmit Pacing Algorithm Test Register	Section 5.39
1E8h	RXPAUSE	Receive Pause Timer Register	Section 5.40
1ECh	TXPAUSE	Transmit Pause Timer Register	Section 5.41
500h	MACADDRLO	MAC Address Low Bytes Register, Used in Receive Address Matching	Section 5.42
504h	MACADDRHI	MAC Address High Bytes Register, Used in Receive Address Matching	Section 5.43
508h	MACINDEX	MAC Index Register	Section 5.44
600h	TX0HDP	Transmit Channel 0 DMA Head Descriptor Pointer Register	Section 5.45
604h	TX1HDP	Transmit Channel 1 DMA Head Descriptor Pointer Register	Section 5.45
608h	TX2HDP	Transmit Channel 2 DMA Head Descriptor Pointer Register	Section 5.45
60Ch	TX3HDP	Transmit Channel 3 DMA Head Descriptor Pointer Register	Section 5.45
610h	TX4HDP	Transmit Channel 4 DMA Head Descriptor Pointer Register	Section 5.45
614h	TX5HDP	Transmit Channel 5 DMA Head Descriptor Pointer Register	Section 5.45
618h	TX6HDP	Transmit Channel 6 DMA Head Descriptor Pointer Register	Section 5.45
61Ch	TX7HDP	Transmit Channel 7 DMA Head Descriptor Pointer Register	Section 5.45
620h	RX0HDP	Receive Channel 0 DMA Head Descriptor Pointer Register	Section 5.46
624h	RX1HDP	Receive Channel 1 DMA Head Descriptor Pointer Register	Section 5.46
628h	RX2HDP	Receive Channel 2 DMA Head Descriptor Pointer Register	Section 5.46
62Ch	RX3HDP	Receive Channel 3 DMA Head Descriptor Pointer Register	Section 5.46
630h	RX4HDP	Receive Channel 4 DMA Head Descriptor Pointer Register	Section 5.46
634h	RX5HDP	Receive Channel 5 DMA Head Descriptor Pointer Register	Section 5.46
638h	RX6HDP	Receive Channel 6 DMA Head Descriptor Pointer Register	Section 5.46
63Ch	RX7HDP	Receive Channel 7 DMA Head Descriptor Pointer Register	Section 5.46
640h	TX0CP	Transmit Channel 0 Completion Pointer Register	Section 5.47
644h	TX1CP	Transmit Channel 1 Completion Pointer Register	Section 5.47
648h	TX2CP	Transmit Channel 2 Completion Pointer Register	Section 5.47
64Ch	TX3CP	Transmit Channel 3 Completion Pointer Register	Section 5.47
650h	TX4CP	Transmit Channel 4 Completion Pointer Register	Section 5.47
654h	TX5CP	Transmit Channel 5 Completion Pointer Register	Section 5.47
658h	TX6CP	Transmit Channel 6 Completion Pointer Register	Section 5.47
65Ch	TX7CP	Transmit Channel 7 Completion Pointer Register	Section 5.47
660h	RX0CP	Receive Channel 0 Completion Pointer Register	Section 5.48
664h	RX1CP	Receive Channel 1 Completion Pointer Register	Section 5.48
668h	RX2CP	Receive Channel 2 Completion Pointer Register	Section 5.48
66Ch	RX3CP	Receive Channel 3 Completion Pointer Register	Section 5.48
670h	RX4CP	Receive Channel 4 Completion Pointer Register	Section 5.48
674h	RX5CP	Receive Channel 5 Completion Pointer Register	Section 5.48
678h	RX6CP	Receive Channel 6 Completion Pointer Register	Section 5.48

Table 25. Ethernet Media Access Controller (EMAC) Registers (continued)

Offset	Acronym	Register Description	Section
67Ch	RX7CP	Receive Channel 7 Completion Pointer Register	Section 5.48
		Network Statistics Registers	
200h	RXGOODFRAMES	Good Receive Frames Register	Section 5.49.1
204h	RXBCASTFRAMES	Broadcast Receive Frames Register	Section 5.49.2
208h	RXMCASTFRAMES	Multicast Receive Frames Register	Section 5.49.3
20Ch	RXPAUSEFRAMES	Pause Receive Frames Register	Section 5.49.4
210h	RXCRCERRORS	Receive CRC Errors Register	Section 5.49.5
214h	RXALIGNCODEERRORS	Receive Alignment/Code Errors Register	Section 5.49.6
218h	RXOVERSIZED	Receive Oversized Frames Register	Section 5.49.7
21Ch	RXJABBER	Receive Jabber Frames Register	Section 5.49.8
220h	RXUNDERSIZED	Receive Undersized Frames Register	Section 5.49.9
224h	RXFRAGMENTS	Receive Frame Fragments Register	Section 5.49.10
228h	RXFILTERED	Filtered Receive Frames Register	Section 5.49.11
22Ch	RXQOSFILTERED	Receive QOS Filtered Frames Register	Section 5.49.12
230h	RXOCTETS	Receive Octet Frames Register	Section 5.49.13
234h	TXGOODFRAMES	Good Transmit Frames Register	Section 5.49.14
238h	TXBCASTFRAMES	Broadcast Transmit Frames Register	Section 5.49.15
23Ch	TXMCASTFRAMES	Multicast Transmit Frames Register	Section 5.49.16
240h	TXPAUSEFRAMES	Pause Transmit Frames Register	Section 5.49.17
244h	TXDEFERRED	Deferred Transmit Frames Register	Section 5.49.18
248h	TXCOLLISION	Transmit Collision Frames Register	Section 5.49.19
24Ch	TXSINGLECOLL	Transmit Single Collision Frames Register	Section 5.49.20
250h	TXMULTICOLL	Transmit Multiple Collision Frames Register	Section 5.49.21
254h	TXEXCESSIVECOLL	Transmit Excessive Collision Frames Register	Section 5.49.22
258h	TXLATECOLL	Transmit Late Collision Frames Register	Section 5.49.23
25Ch	TXUNDERRUN	Transmit Underrun Error Register	Section 5.49.24
260h	TXCARRIERSENSE	Transmit Carrier Sense Errors Register	Section 5.49.25
264h	TXOCTETS	Transmit Octet Frames Register	Section 5.49.26
268h	FRAME64	Transmit and Receive 64 Octet Frames Register	Section 5.49.27
26Ch	FRAME65T127	Transmit and Receive 65 to 127 Octet Frames Register	Section 5.49.28
270h	FRAME128T255	Transmit and Receive 128 to 255 Octet Frames Register	Section 5.49.29
274h	FRAME256T511	Transmit and Receive 256 to 511 Octet Frames Register	Section 5.49.30
278h	FRAME512T1023	Transmit and Receive 512 to 1023 Octet Frames Register	Section 5.49.31
27Ch	FRAME1024TUP	Transmit and Receive 1024 to RXMAXLEN Octet Frames Register	Section 5.49.32
280h	NETOCTETS	Network Octet Frames Register	Section 5.49.33
284h	RXSOFOVERRUNS	Receive FIFO or DMA Start of Frame Overruns Register	Section 5.49.34
288h	RXMOFOVERRUNS	Receive FIFO or DMA Middle of Frame Overruns Register	Section 5.49.35
28Ch	RXDMAOVERRUNS	Receive DMA Overruns Register	Section 5.49.36

5.1 Transmit Identification and Version Register (TXIDVER)

The transmit identification and version register (TXIDVER) is shown in Figure 27 and described in Table 26.

Figure 27. Transmit Identification and Version Register (TXIDVER)

31	16
TXIDENT	
R-0Ch	

15	8	7	0
TXMAJORVER		TXMINORVER	
R-0Ah		R-07h	

LEGEND: R = Read only; -*n* = value after reset

Table 26. Transmit Identification and Version Register (TXIDVER) Field Descriptions

Bit	Field	Value	Description
31-16	TXIDENT		Transmit identification value.
		Ch	Current transmit identification value.
15-8	TXMAJORVER		Transmit major version value. Revisions are indicated by a revision code taking the format TXMAJORVER.TXMINORVER.
		Ah	Current transmit major version value.
7-0	TXMINORVER		Transmit minor version value. Revisions are indicated by a revision code taking the format TXMAJORVER.TXMINORVER.
		7h	Current transmit minor version value.

5.2 Transmit Control Register (TXCONTROL)

The transmit control register (TXCONTROL) is shown in Figure 28 and described in Table 27.

Figure 28. Transmit Control Register (TXCONTROL)

31	16
Reserved	
R-0	

15	1	0
Reserved		TXEN
R-0		R/W-0

LEGEND: R = Read only; R/W = Read/Write; -*n* = value after reset

Table 27. Transmit Control Register (TXCONTROL) Field Descriptions

Bit	Field	Value	Description
31-1	Reserved	0	Reserved
0	TXEN		Transmit enable
		0	Transmit is disabled.
		1	Transmit is enabled.

5.3 Transmit Teardown Register (TXTEARDOWN)

The transmit teardown register (TXTEARDOWN) is shown in Figure 29 and described in Table 28.

Figure 29. Transmit Teardown Register (TXTEARDOWN)

31	16
Reserved	
R-0	

15	3	2	0
Reserved		TXTDNCH	
R-0		R/W-0	

LEGEND: R = Read only; R/W = Read/Write; -*n* = value after reset

Table 28. Transmit Teardown Register (TXTEARDOWN) Field Descriptions

Bit	Field	Value	Description
31-3	Reserved	0	Reserved
2-0	TXTDNCH	0-7h	Transmit teardown channel. The transmit channel teardown is commanded by writing the encoded value of the transmit channel to be torn down. The teardown register is read as 0.
		0	Teardown transmit channel 0
		1h	Teardown transmit channel 1
		2h	Teardown transmit channel 2
		3h	Teardown transmit channel 3
		4h	Teardown transmit channel 4
		5h	Teardown transmit channel 5
		6h	Teardown transmit channel 6
		7h	Teardown transmit channel 7

5.4 Receive Identification and Version Register (RXIDVER)

The receive identification and version register (RXIDVER) is shown in Figure 30 and described in Table 29.

Figure 30. Receive Identification and Version Register (RXIDVER)

31	16
RXIDENT	
R-0Ch	

15	8	7	0
RXMAJORVER		RXMINORVER	
R-0Ah		R-07h	

LEGEND: R = Read only; -*n* = value after reset

Table 29. Receive Identification and Version Register (RXIDVER) Field Descriptions

Bit	Field	Value	Description
31-16	RXIDENT		Receive identification value.
		Ch	Current receive identification value.
15-8	RXMAJORVER		Receive major version value. Revisions are indicated by a revision code taking the format RXMAJORVER.RXMINORVER.
		Ah	Current receive major version value.
7-0	RXMINORVER		Receive minor version value. Revisions are indicated by a revision code taking the format RXMAJORVER.RXMINORVER.
		7h	Current receive minor version value.

5.5 Receive Control Register (RXCONTROL)

The receive control register (RXCONTROL) is shown in Figure 31 and described in Table 30.

Figure 31. Receive Control Register (RXCONTROL)

31	16
Reserved	
R-0	

15	1	0
Reserved		RXEN
R-0		R/W-0

LEGEND: R = Read only; R/W = Read/Write; -*n* = value after reset

Table 30. Receive Control Register (RXCONTROL) Field Descriptions

Bit	Field	Value	Description
31-1	Reserved	0	Reserved
0	RXEN		Receive enable
		0	Receive is disabled.
		1	Receive is enabled.

5.6 Receive Teardown Register (RXTEARDOWN)

The receive teardown register (RXTEARDOWN) is shown in Figure 32 and described in Table 31.

Figure 32. Receive Teardown Register (RXTEARDOWN)

31	16
Reserved	
R-0	

15	3	2	0
Reserved		RXTDNCH	
R-0		R/W-0	

LEGEND: R = Read only; R/W = Read/Write; -*n* = value after reset

Table 31. Receive Teardown Register (RXTEARDOWN) Field Descriptions

Bit	Field	Value	Description
31-3	Reserved	0	Reserved
2-0	RXTDNCH	0-7h	Receive teardown channel. The receive channel teardown is commanded by writing the encoded value of the receive channel to be torn down. The teardown register is read as 0.
		0	Teardown receive channel 0
		1h	Teardown receive channel 1
		2h	Teardown receive channel 2
		3h	Teardown receive channel 3
		4h	Teardown receive channel 4
		5h	Teardown receive channel 5
		6h	Teardown receive channel 6
		7h	Teardown receive channel 7

5.7 Transmit Interrupt Status (Unmasked) Register (TXINTSTATRAW)

The transmit interrupt status (unmasked) register (TXINTSTATRAW) is shown in Figure 33 and described in Table 32.

Figure 33. Transmit Interrupt Status (Unmasked) Register (TXINTSTATRAW)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
R-0															

15	14	13	12	11	10	9	8
Reserved							
R-0							

7	6	5	4	3	2	1	0
TX7PEND	TX6PEND	TX5PEND	TX4PEND	TX3PEND	TX2PEND	TX1PEND	TX0PEND
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

Table 32. Transmit Interrupt Status (Unmasked) Register (TXINTSTATRAW) Field Descriptions

Bit	Field	Value	Description
31-8	Reserved	0	Reserved
7	TX7PEND	0-1	TX7PEND raw interrupt read (before mask)
6	TX6PEND	0-1	TX6PEND raw interrupt read (before mask)
5	TX5PEND	0-1	TX5PEND raw interrupt read (before mask)
4	TX4PEND	0-1	TX4PEND raw interrupt read (before mask)
3	TX3PEND	0-1	TX3PEND raw interrupt read (before mask)
2	TX2PEND	0-1	TX2PEND raw interrupt read (before mask)
1	TX1PEND	0-1	TX1PEND raw interrupt read (before mask)
0	TX0PEND	0-1	TX0PEND raw interrupt read (before mask)

5.8 Transmit Interrupt Status (Masked) Register (TXINTSTATMASKED)

The transmit interrupt status (masked) register (TXINTSTATMASKED) is shown in Figure 34 and described in Table 33.

Figure 34. Transmit Interrupt Status (Masked) Register (TXINTSTATMASKED)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
R-0															

15	14	13	12	11	10	9	8
Reserved							
R-0							

7	6	5	4	3	2	1	0
TX7PEND	TX6PEND	TX5PEND	TX4PEND	TX3PEND	TX2PEND	TX1PEND	TX0PEND
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

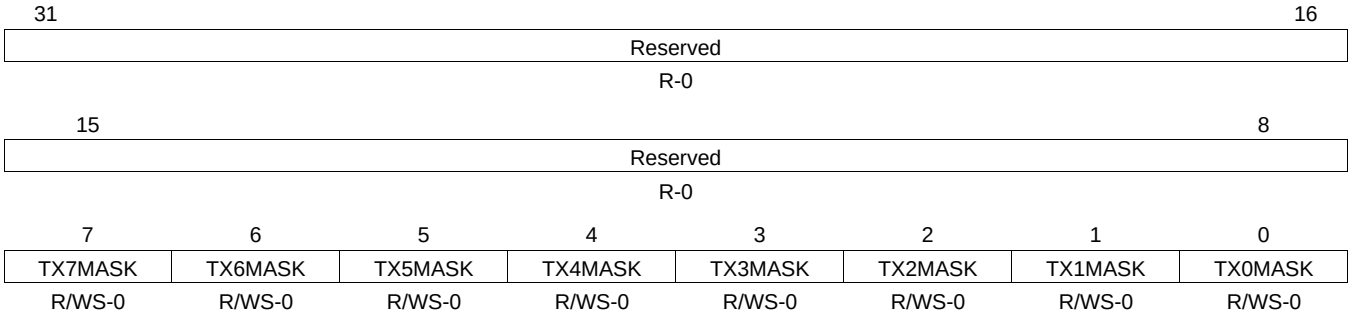
Table 33. Transmit Interrupt Status (Masked) Register (TXINTSTATMASKED) Field Descriptions

Bit	Field	Value	Description
31-8	Reserved	0	Reserved
7	TX7PEND	0-1	TX7PEND masked interrupt read
6	TX6PEND	0-1	TX6PEND masked interrupt read
5	TX5PEND	0-1	TX5PEND masked interrupt read
4	TX4PEND	0-1	TX4PEND masked interrupt read
3	TX3PEND	0-1	TX3PEND masked interrupt read
2	TX2PEND	0-1	TX2PEND masked interrupt read
1	TX1PEND	0-1	TX1PEND masked interrupt read
0	TX0PEND	0-1	TX0PEND masked interrupt read

5.9 Transmit Interrupt Mask Set Register (TXINTMASKSET)

The transmit interrupt mask set register (TXINTMASKSET) is shown in Figure 35 and described in Table 34.

Figure 35. Transmit Interrupt Mask Set Register (TXINTMASKSET)



LEGEND: R = Read only; R/W = Read/Write; WS = Write 1 to set, write of 0 has no effect; -*n* = value after reset

Table 34. Transmit Interrupt Mask Set Register (TXINTMASKSET) Field Descriptions

Bit	Field	Value	Description
31-8	Reserved	0	Reserved
7	TX7MASK	0-1	Transmit channel 7 interrupt mask set bit. Write 1 to enable interrupt, a write of 0 has no effect.
6	TX6MASK	0-1	Transmit channel 6 interrupt mask set bit. Write 1 to enable interrupt, a write of 0 has no effect.
5	TX5MASK	0-1	Transmit channel 5 interrupt mask set bit. Write 1 to enable interrupt, a write of 0 has no effect.
4	TX4MASK	0-1	Transmit channel 4 interrupt mask set bit. Write 1 to enable interrupt, a write of 0 has no effect.
3	TX3MASK	0-1	Transmit channel 3 interrupt mask set bit. Write 1 to enable interrupt, a write of 0 has no effect.
2	TX2MASK	0-1	Transmit channel 2 interrupt mask set bit. Write 1 to enable interrupt, a write of 0 has no effect.
1	TX1MASK	0-1	Transmit channel 1 interrupt mask set bit. Write 1 to enable interrupt, a write of 0 has no effect.
0	TX0MASK	0-1	Transmit channel 0 interrupt mask set bit. Write 1 to enable interrupt, a write of 0 has no effect.

5.10 Transmit Interrupt Mask Clear Register (TXINTMASKCLEAR)

The transmit interrupt mask clear register (TXINTMASKCLEAR) is shown in Figure 36 and described in Table 35.

Figure 36. Transmit Interrupt Mask Clear Register (TXINTMASKCLEAR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
R-0															

15	14	13	12	11	10	9	8
Reserved							
R-0							

7	6	5	4	3	2	1	0
TX7MASK	TX6MASK	TX5MASK	TX4MASK	TX3MASK	TX2MASK	TX1MASK	TX0MASK
R/WC-0	R/WC-0	R/WC-0	R/WC-0	R/WC-0	R/WC-0	R/WC-0	R/WC-0

LEGEND: R = Read only; R/W = Read/Write; WC = Write 1 to clear, write of 0 has no effect; -*n* = value after reset

Table 35. Transmit Interrupt Mask Clear Register (TXINTMASKCLEAR) Field Descriptions

Bit	Field	Value	Description
31-8	Reserved	0	Reserved
7	TX7MASK	0-1	Transmit channel 7 interrupt mask clear bit. Write 1 to disable interrupt, a write of 0 has no effect.
6	TX6MASK	0-1	Transmit channel 6 interrupt mask clear bit. Write 1 to disable interrupt, a write of 0 has no effect.
5	TX5MASK	0-1	Transmit channel 5 interrupt mask clear bit. Write 1 to disable interrupt, a write of 0 has no effect.
4	TX4MASK	0-1	Transmit channel 4 interrupt mask clear bit. Write 1 to disable interrupt, a write of 0 has no effect.
3	TX3MASK	0-1	Transmit channel 3 interrupt mask clear bit. Write 1 to disable interrupt, a write of 0 has no effect.
2	TX2MASK	0-1	Transmit channel 2 interrupt mask clear bit. Write 1 to disable interrupt, a write of 0 has no effect.
1	TX1MASK	0-1	Transmit channel 1 interrupt mask clear bit. Write 1 to disable interrupt, a write of 0 has no effect.
0	TX0MASK	0-1	Transmit channel 0 interrupt mask clear bit. Write 1 to disable interrupt, a write of 0 has no effect.

5.11 MAC Input Vector Register (MACINVECTOR)

The MAC input vector register (MACINVECTOR) is shown in Figure 37 and described in Table 36.

Figure 37. MAC Input Vector Register (MACINVECTOR)

31	30	29	18	17	16
USERINT	LINKINT	Reserved		HOSTPEND	STATPEND
R-0	R-0	R-0		R-0	R-0

15	8	7	0
RXPEND		TXPEND	
R-0		R-0	

LEGEND: R = Read only; -*n* = value after reset

Table 36. MAC Input Vector Register (MACINVECTOR) Field Descriptions

Bit	Field	Value	Description
31	USERINT	0-1	MDIO module user interrupt (USERINT) pending status bit.
30	LINKINT	0-1	MDIO module link change interrupt (LINKINT) pending status bit.
29-18	Reserved	0	Reserved
17	HOSTPEND	0-1	EMAC module host error interrupt (HOSTPEND) pending status bit.
16	STATPEND	0-1	EMAC module statistics interrupt (STATPEND) pending status bit.
15-8	RXPEND	0-FFh	Receive channels 0-7 interrupt (RX*n*PEND) pending status bit. Bit 8 is receive channel 0.
7-0	TXPEND	0-FFh	Transmit channels 0-7 interrupt (TX*n*PEND) pending status bit. Bit 0 is transmit channel 0.

5.12 Receive Interrupt Status (Unmasked) Register (RXINTSTATRAW)

The receive interrupt status (unmasked) register (RXINTSTATRAW) is shown in Figure 38 and described in Table 37.

Figure 38. Receive Interrupt Status (Unmasked) Register (RXINTSTATRAW)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
R-0															

15	14	13	12	11	10	9	8
Reserved							
R-0							

7	6	5	4	3	2	1	0
RX7PEND	RX6PEND	RX5PEND	RX4PEND	RX3PEND	RX2PEND	RX1PEND	RX0PEND
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

Table 37. Receive Interrupt Status (Unmasked) Register (RXINTSTATRAW) Field Descriptions

Bit	Field	Value	Description
31-8	Reserved	0	Reserved
7	RX7PEND	0-1	RX7PEND raw interrupt read (before mask)
6	RX6PEND	0-1	RX6PEND raw interrupt read (before mask)
5	RX5PEND	0-1	RX5PEND raw interrupt read (before mask)
4	RX4PEND	0-1	RX4PEND raw interrupt read (before mask)
3	RX3PEND	0-1	RX3PEND raw interrupt read (before mask)
2	RX2PEND	0-1	RX2PEND raw interrupt read (before mask)
1	RX1PEND	0-1	RX1PEND raw interrupt read (before mask)
0	RX0PEND	0-1	RX0PEND raw interrupt read (before mask)

5.13 Receive Interrupt Status (Masked) Register (RXINTSTATMASKED)

The receive interrupt status (masked) register (RXINTSTATMASKED) is shown in Figure 39 and described in Table 38.

Figure 39. Receive Interrupt Status (Masked) Register (RXINTSTATMASKED)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
R-0															

15	14	13	12	11	10	9	8
Reserved							
R-0							

7	6	5	4	3	2	1	0
RX7PEND	RX6PEND	RX5PEND	RX4PEND	RX3PEND	RX2PEND	RX1PEND	RX0PEND
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

Table 38. Receive Interrupt Status (Masked) Register (RXINTSTATMASKED) Field Descriptions

Bit	Field	Value	Description
31-8	Reserved	0	Reserved
7	RX7PEND	0-1	RX7PEND masked interrupt read
6	RX6PEND	0-1	RX6PEND masked interrupt read
5	RX5PEND	0-1	RX5PEND masked interrupt read
4	RX4PEND	0-1	RX4PEND masked interrupt read
3	RX3PEND	0-1	RX3PEND masked interrupt read
2	RX2PEND	0-1	RX2PEND masked interrupt read
1	RX1PEND	0-1	RX1PEND masked interrupt read
0	RX0PEND	0-1	RX0PEND masked interrupt read

5.14 Receive Interrupt Mask Set Register (RXINTMASKSET)

The receive interrupt mask set register (RXINTMASKSET) is shown in Figure 40 and described in Table 39.

Figure 40. Receive Interrupt Mask Set Register (RXINTMASKSET)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
R-0															

15	14	13	12	11	10	9	8
Reserved							
R-0							

7	6	5	4	3	2	1	0
RX7MASK	RX6MASK	RX5MASK	RX4MASK	RX3MASK	RX2MASK	RX1MASK	RX0MASK
R/WS-0	R/WS-0	R/WS-0	R/WS-0	R/WS-0	R/WS-0	R/WS-0	R/WS-0

LEGEND: R = Read only; R/W = Read/Write; WS = Write 1 to set, write of 0 has no effect; -*n* = value after reset

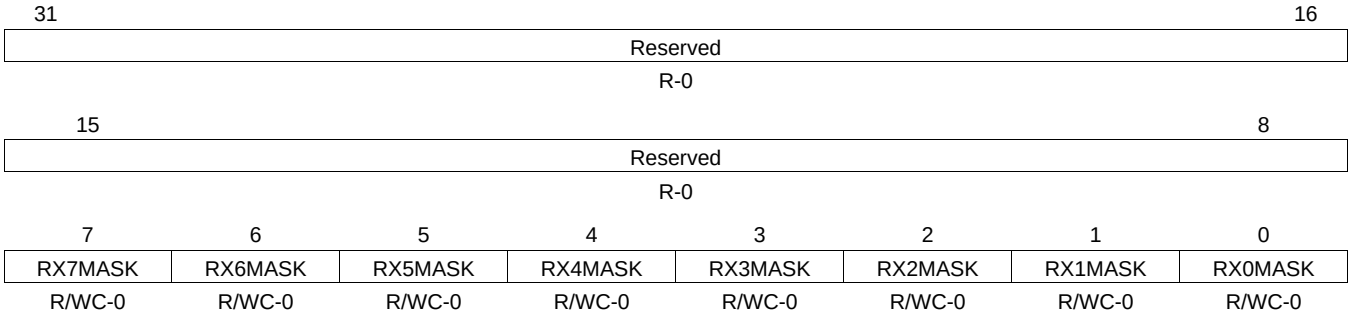
Table 39. Receive Interrupt Mask Set Register (RXINTMASKSET) Field Descriptions

Bit	Field	Value	Description
31-8	Reserved	0	Reserved
7	RX7MASK	0-1	Receive channel 7 mask set bit. Write 1 to enable interrupt, a write of 0 has no effect.
6	RX6MASK	0-1	Receive channel 6 mask set bit. Write 1 to enable interrupt, a write of 0 has no effect.
5	RX5MASK	0-1	Receive channel 5 mask set bit. Write 1 to enable interrupt, a write of 0 has no effect.
4	RX4MASK	0-1	Receive channel 4 mask set bit. Write 1 to enable interrupt, a write of 0 has no effect.
3	RX3MASK	0-1	Receive channel 3 mask set bit. Write 1 to enable interrupt, a write of 0 has no effect.
2	RX2MASK	0-1	Receive channel 2 mask set bit. Write 1 to enable interrupt, a write of 0 has no effect.
1	RX1MASK	0-1	Receive channel 1 mask set bit. Write 1 to enable interrupt, a write of 0 has no effect.
0	RX0MASK	0-1	Receive channel 0 mask set bit. Write 1 to enable interrupt, a write of 0 has no effect.

5.15 Receive Interrupt Mask Clear Register (RXINTMASKCLEAR)

The receive interrupt mask clear register (RXINTMASKCLEAR) is shown in Figure 41 and described in Table 40.

Figure 41. Receive Interrupt Mask Clear Register (RXINTMASKCLEAR)



LEGEND: R = Read only; R/W = Read/Write; WC = Write 1 to clear, write of 0 has no effect; -*n* = value after reset

Table 40. Receive Interrupt Mask Clear Register (RXINTMASKCLEAR) Field Descriptions

Bit	Field	Value	Description
31-8	Reserved	0	Reserved
7	RX7MASK	0-1	Receive channel 7 mask clear bit. Write 1 to disable interrupt, a write of 0 has no effect.
6	RX6MASK	0-1	Receive channel 6 mask clear bit. Write 1 to disable interrupt, a write of 0 has no effect.
5	RX5MASK	0-1	Receive channel 5 mask clear bit. Write 1 to disable interrupt, a write of 0 has no effect.
4	RX4MASK	0-1	Receive channel 4 mask clear bit. Write 1 to disable interrupt, a write of 0 has no effect.
3	RX3MASK	0-1	Receive channel 3 mask clear bit. Write 1 to disable interrupt, a write of 0 has no effect.
2	RX2MASK	0-1	Receive channel 2 mask clear bit. Write 1 to disable interrupt, a write of 0 has no effect.
1	RX1MASK	0-1	Receive channel 1 mask clear bit. Write 1 to disable interrupt, a write of 0 has no effect.
0	RX0MASK	0-1	Receive channel 0 mask clear bit. Write 1 to disable interrupt, a write of 0 has no effect.

5.16 MAC Interrupt Status (Unmasked) Register (MACINTSTATRAW)

The MAC interrupt status (unmasked) register (MACINTSTATRAW) is shown in Figure 42 and described in Table 41.

Figure 42. MAC Interrupt Status (Unmasked) Register (MACINTSTATRAW)

31	16
Reserved	
R-0	

15	2	1	0
Reserved		HOSTPEND	STATPEND
R-0		R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

Table 41. MAC Interrupt Status (Unmasked) Register (MACINTSTATRAW) Field Descriptions

Bit	Field	Value	Description
31-2	Reserved	0	Reserved
1	HOSTPEND	0-1	Host pending interrupt (HOSTPEND); raw interrupt read (before mask).
0	STATPEND	0-1	Statistics pending interrupt (STATPEND); raw interrupt read (before mask).

5.17 MAC Interrupt Status (Masked) Register (MACINTSTATMASKED)

The MAC interrupt status (masked) register (MACINTSTATMASKED) is shown in Figure 43 and described in Table 42.

Figure 43. MAC Interrupt Status (Masked) Register (MACINTSTATMASKED)

31	16
Reserved	
R-0	

15	2	1	0
Reserved		HOSTPEND	STATPEND
R-0		R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

Table 42. MAC Interrupt Status (Masked) Register (MACINTSTATMASKED) Field Descriptions

Bit	Field	Value	Description
31-2	Reserved	0	Reserved
1	HOSTPEND	0-1	Host pending interrupt (HOSTPEND); masked interrupt read.
0	STATPEND	0-1	Statistics pending interrupt (STATPEND); masked interrupt read.

5.18 MAC Interrupt Mask Set Register (MACINTMASKSET)

The MAC interrupt mask set register (MACINTMASKSET) is shown in Figure 44 and described in Table 43.

Figure 44. MAC Interrupt Mask Set Register (MACINTMASKSET)

31	16
Reserved	
R-0	

15	2	1	0
Reserved		HOSTMASK	STATMASK
R-0		R/WS-0	R/WS-0

LEGEND: R = Read only; R/W = Read/Write; WS = Write 1 to set, write of 0 has no effect; -*n* = value after reset

Table 43. MAC Interrupt Mask Set Register (MACINTMASKSET) Field Descriptions

Bit	Field	Value	Description
31-2	Reserved	0	Reserved
1	HOSTMASK	0-1	Host error interrupt mask set bit. Write 1 to enable interrupt, a write of 0 has no effect.
0	STATMASK	0-1	Statistics interrupt mask set bit. Write 1 to enable interrupt, a write of 0 has no effect.

5.19 MAC Interrupt Mask Clear Register (MACINTMASKCLEAR)

The MAC interrupt mask clear register (MACINTMASKCLEAR) is shown in Figure 45 and described in Table 44.

Figure 45. MAC Interrupt Mask Clear Register (MACINTMASKCLEAR)

31	16
Reserved	
R-0	

15	2	1	0
Reserved		HOSTMASK	STATMASK
R-0		R/WC-0	R/WC-0

LEGEND: R = Read only; R/W = Read/Write; WC = Write 1 to clear, write of 0 has no effect; -*n* = value after reset

Table 44. MAC Interrupt Mask Clear Register (MACINTMASKCLEAR) Field Descriptions

Bit	Field	Value	Description
31-2	Reserved	0	Reserved
1	HOSTMASK	0-1	Host error interrupt mask clear bit. Write 1 to disable interrupt, a write of 0 has no effect.
0	STATMASK	0-1	Statistics interrupt mask clear bit. Write 1 to disable interrupt, a write of 0 has no effect.

5.20 Receive Multicast/Broadcast/Promiscuous Channel Enable Register (RXMBPENABLE)

The receive multicast/broadcast/promiscuous channel enable register (RXMBPENABLE) is shown in Figure 46 and described in Table 45.

Figure 46. Receive Multicast/Broadcast/Promiscuous Channel Enable Register (RXMBPENABLE)

31	30	29	28	27	25	24
Reserved	RXPASSCRC	RXQOSEN	RXNOCHAIN	Reserved		RXCMFEN
R-0	R/W-0	R/W-0	R/W-0	R-0		R/W-0

23	22	21	20	19	18	16
RXCSFEN	RXCEFEN	RXCAFEN	Reserved		RXPROMCH	
R/W-0	R/W-0	R/W-0	R-0		R/W-0	

15	14	13	12	11	10	8
Reserved		RXBROADEN	Reserved		RXBROADCH	
R-0		R/W-0	R-0		R/W-0	

7	6	5	4	3	2	0
Reserved		RXMULTEN	Reserved		RXMULTCH	
R-0		R/W-0	R-0		R/W-0	

LEGEND: R = Read only; R/W = Read/Write; -*n* = value after reset

Table 45. Receive Multicast/Broadcast/Promiscuous Channel Enable Register (RXMBPENABLE) Field Descriptions

Bit	Field	Value	Description
31	Reserved	0	Reserved
30	RXPASSCRC		Pass receive CRC enable bit
		0	Received CRC is discarded for all channels and is not included in the buffer descriptor packet length field.
		1	Received CRC is transferred to memory for all channels and is included in the buffer descriptor packet length.
29	RXQOSEN		Receive quality of service enable bit
		0	Receive QOS is disabled.
		1	Receive QOS is enabled.
28	RXNOCHAIN		Receive no buffer chaining bit
		0	Received frames can span multiple buffers.
		1	The Receive DMA controller transfers each frame into a single buffer, regardless of the frame or buffer size. All remaining frame data after the first buffer is discarded. The buffer descriptor buffer length field will contain the entire frame byte count (up to 65535 bytes).
27-25	Reserved	0	Reserved
24	RXCMFEN		Receive copy MAC control frames enable bit. Enables MAC control frames to be transferred to memory. MAC control frames are normally acted upon (if enabled), but not copied to memory. MAC control frames that are pause frames will be acted upon if enabled in MACCONTROL, regardless of the value of RXCMFEN. Frames transferred to memory due to RXCMFEN will have the CONTROL bit set in their EOP buffer descriptor.
		0	MAC control frames are filtered (but acted upon if enabled).
		1	MAC control frames are transferred to memory.
23	RXCSFEN		Receive copy short frames enable bit. Enables frames or fragments shorter than 64 bytes to be copied to memory. Frames transferred to memory due to RXCSFEN will have the FRAGMENT or UNDERSIZE bit set in their EOP buffer descriptor. Fragments are short frames that contain CRC / align / code errors and undersized are short frames without errors.
		0	Short frames are filtered.
		1	Short frames are transferred to memory.

Table 45. Receive Multicast/Broadcast/Promiscuous Channel Enable Register (RXMBPENABLE) Field Descriptions (continued)

Bit	Field	Value	Description
22	RXCEFEN		Receive copy error frames enable bit. Enables frames containing errors to be transferred to memory. The appropriate error bit will be set in the frame EOP buffer descriptor.
		0	Frames containing errors are filtered.
		1	Frames containing errors are transferred to memory.
21	RXCAFEN		Receive copy all frames enable bit. Enables frames that do not address match (includes multicast frames that do not hash match) to be transferred to the promiscuous channel selected by RXPROMCH bits. Such frames will be marked with the NOMATCH bit in their EOP buffer descriptor.
		0	Frames that do not address match are filtered.
		1	Frames that do not address match are transferred to the promiscuous channel selected by RXPROMCH bits.
20-19	Reserved	0	Reserved
18-16	RXPROMCH	0-7h	Receive promiscuous channel select
		0	Select channel 0 to receive promiscuous frames
		1h	Select channel 1 to receive promiscuous frames
		2h	Select channel 2 to receive promiscuous frames
		3h	Select channel 3 to receive promiscuous frames
		4h	Select channel 4 to receive promiscuous frames
		5h	Select channel 5 to receive promiscuous frames
		6h	Select channel 6 to receive promiscuous frames
		7h	Select channel 7 to receive promiscuous frames
15-14	Reserved	0	Reserved
13	RXBROADEN		Receive broadcast enable. Enable received broadcast frames to be copied to the channel selected by RXBROADCH bits.
		0	Broadcast frames are filtered.
		1	Broadcast frames are copied to the channel selected by RXBROADCH bits.
12-11	Reserved	0	Reserved
10-8	RXBROADCH	0-7h	Receive broadcast channel select
		0	Select channel 0 to receive broadcast frames
		1h	Select channel 1 to receive broadcast frames
		2h	Select channel 2 to receive broadcast frames
		3h	Select channel 3 to receive broadcast frames
		4h	Select channel 4 to receive broadcast frames
		5h	Select channel 5 to receive broadcast frames
		6h	Select channel 6 to receive broadcast frames
		7h	Select channel 7 to receive broadcast frames
7-6	Reserved	0	Reserved
5	RXMULTEN		RX multicast enable. Enable received hash matching multicast frames to be copied to the channel selected by RXMULTCH bits.
		0	Multicast frames are filtered.
		1	Multicast frames are copied to the channel selected by RXMULTCH bits.
4-3	Reserved	0	Reserved

Table 45. Receive Multicast/Broadcast/Promiscuous Channel Enable Register (RXMBPENABLE) Field Descriptions (continued)

Bit	Field	Value	Description
2-0	RXMULTCH	0-7h	Receive multicast channel select
		0	Select channel 0 to receive multicast frames
		1h	Select channel 1 to receive multicast frames
		2h	Select channel 2 to receive multicast frames
		3h	Select channel 3 to receive multicast frames
		4h	Select channel 4 to receive multicast frames
		5h	Select channel 5 to receive multicast frames
		6h	Select channel 6 to receive multicast frames
		7h	Select channel 7 to receive multicast frames

5.21 Receive Unicast Enable Set Register (RXUNICASTSET)

The receive unicast enable set register (RXUNICASTSET) is shown in Figure 47 and described in Table 46.

Figure 47. Receive Unicast Enable Set Register (RXUNICASTSET)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
R-0															

15	14	13	12	11	10	9	8
Reserved							
R-0							

7	6	5	4	3	2	1	0
RXCH7EN	RXCH6EN	RXCH5EN	RXCH4EN	RXCH3EN	RXCH2EN	RXCH1EN	RXCH0EN
R/WS-0	R/WS-0	R/WS-0	R/WS-0	R/WS-0	R/WS-0	R/WS-0	R/WS-0

LEGEND: R = Read only; R/W = Read/Write; WS = Write 1 to set, write of 0 has no effect; -*n* = value after reset

Table 46. Receive Unicast Enable Set Register (RXUNICASTSET) Field Descriptions

Bit	Field	Value	Description
31-8	Reserved	0	Reserved
7	RXCH7EN	0-1	Receive channel 7 unicast enable set bit. Write 1 to set the enable, a write of 0 has no effect. May be read.
6	RXCH6EN	0-1	Receive channel 6 unicast enable set bit. Write 1 to set the enable, a write of 0 has no effect. May be read.
5	RXCH5EN	0-1	Receive channel 5 unicast enable set bit. Write 1 to set the enable, a write of 0 has no effect. May be read.
4	RXCH4EN	0-1	Receive channel 4 unicast enable set bit. Write 1 to set the enable, a write of 0 has no effect. May be read.
3	RXCH3EN	0-1	Receive channel 3 unicast enable set bit. Write 1 to set the enable, a write of 0 has no effect. May be read.
2	RXCH2EN	0-1	Receive channel 2 unicast enable set bit. Write 1 to set the enable, a write of 0 has no effect. May be read.
1	RXCH1EN	0-1	Receive channel 1 unicast enable set bit. Write 1 to set the enable, a write of 0 has no effect. May be read.
0	RXCH0EN	0-1	Receive channel 0 unicast enable set bit. Write 1 to set the enable, a write of 0 has no effect. May be read.

5.22 Receive Unicast Clear Register (RXUNICASTCLEAR)

The receive unicast clear register (RXUNICASTCLEAR) is shown in Figure 48 and described in Table 47.

Figure 48. Receive Unicast Clear Register (RXUNICASTCLEAR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
R-0															

15	14	13	12	11	10	9	8
Reserved							
R-0							

7	6	5	4	3	2	1	0
RXCH7EN	RXCH6EN	RXCH5EN	RXCH4EN	RXCH3EN	RXCH2EN	RXCH1EN	RXCH0EN
R/WC-0	R/WC-0	R/WC-0	R/WC-0	R/WC-0	R/WC-0	R/WC-0	R/WC-0

LEGEND: R = Read only; R/W = Read/Write; WC = Write 1 to clear, write of 0 has no effect; -*n* = value after reset

Table 47. Receive Unicast Clear Register (RXUNICASTCLEAR) Field Descriptions

Bit	Field	Value	Description
31-8	Reserved	0	Reserved
7	RXCH7EN	0-1	Receive channel 7 unicast enable clear bit. Write 1 to clear the enable, a write of 0 has no effect.
6	RXCH6EN	0-1	Receive channel 6 unicast enable clear bit. Write 1 to clear the enable, a write of 0 has no effect.
5	RXCH5EN	0-1	Receive channel 5 unicast enable clear bit. Write 1 to clear the enable, a write of 0 has no effect.
4	RXCH4EN	0-1	Receive channel 4 unicast enable clear bit. Write 1 to clear the enable, a write of 0 has no effect.
3	RXCH3EN	0-1	Receive channel 3 unicast enable clear bit. Write 1 to clear the enable, a write of 0 has no effect.
2	RXCH2EN	0-1	Receive channel 2 unicast enable clear bit. Write 1 to clear the enable, a write of 0 has no effect.
1	RXCH1EN	0-1	Receive channel 1 unicast enable clear bit. Write 1 to clear the enable, a write of 0 has no effect.
0	RXCH0EN	0-1	Receive channel 0 unicast enable clear bit. Write 1 to clear the enable, a write of 0 has no effect.

5.23 Receive Maximum Length Register (RXMAXLEN)

The receive maximum length register (RXMAXLEN) is shown in Figure 49 and described in Table 48.

Figure 49. Receive Maximum Length Register (RXMAXLEN)

31	16
Reserved	
R-0	
15	0
RXMAXLEN	
R/W-5EEh	

LEGEND: R = Read only; R/W = Read/Write; -*n* = value after reset

Table 48. Receive Maximum Length Register (RXMAXLEN) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved	0	Reserved
15-0	RXMAXLEN	0-FFFFh	Receive maximum frame length. These bits determine the maximum length of a received frame. The reset value is 5EEh (1518). Frames with byte counts greater than RXMAXLEN are long frames. Long frames with no errors are oversized frames. Long frames with CRC, code, or alignment error are jabber frames.

5.24 Receive Buffer Offset Register (RXBUFFEROFFSET)

The receive buffer offset register (RXBUFFEROFFSET) is shown in Figure 50 and described in Table 49.

Figure 50. Receive Buffer Offset Register (RXBUFFEROFFSET)

31	16
Reserved	
R-0	
15	0
RXBUFFEROFFSET	
R/W-0	

LEGEND: R = Read only; R/W = Read/Write; -*n* = value after reset

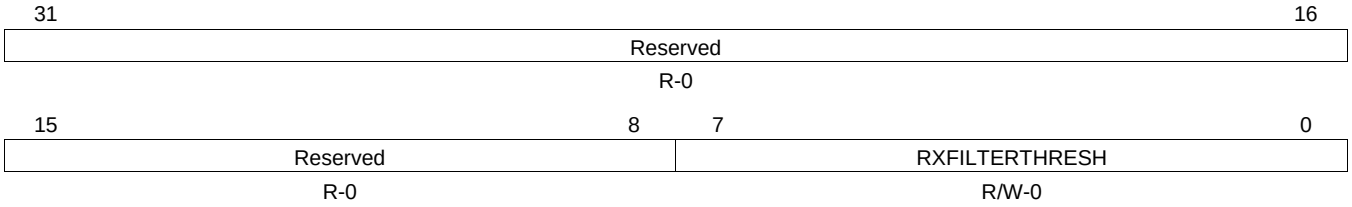
Table 49. Receive Buffer Offset Register (RXBUFFEROFFSET) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved	0	Reserved
15-0	RXBUFFEROFFSET	0-FFFFh	Receive buffer offset value. These bits are written by the EMAC into each frame SOP buffer descriptor Buffer Offset field. The frame data begins after the RXBUFFEROFFSET value of bytes. A value of 0 indicates that there are no unused bytes at the beginning of the data, and that valid data begins on the first byte of the buffer. A value of Fh (15) indicates that the first 15 bytes of the buffer are to be ignored by the EMAC and that valid buffer data starts on byte 16 of the buffer. This value is used for all channels.

5.25 Receive Filter Low Priority Frame Threshold Register (RXFILTERLOWTHRESH)

The receive filter low priority frame threshold register (RXFILTERLOWTHRESH) is shown in Figure 51 and described in Table 50.

Figure 51. Receive Filter Low Priority Frame Threshold Register (RXFILTERLOWTHRESH)



LEGEND: R = Read only; R/W = Read/Write; -*n* = value after reset

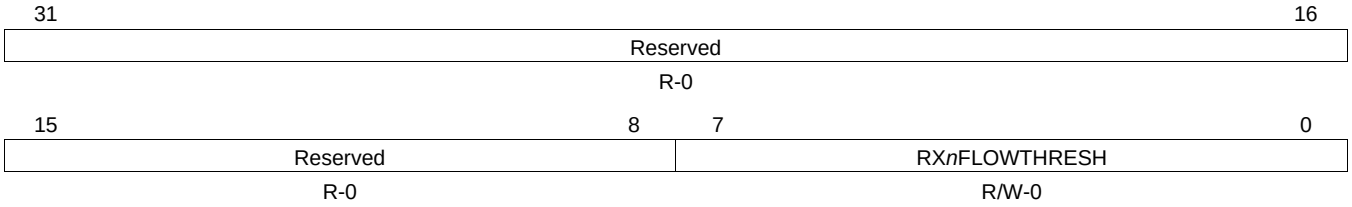
Table 50. Receive Filter Low Priority Frame Threshold Register (RXFILTERLOWTHRESH) Field Descriptions

Bit	Field	Value	Description
31-8	Reserved	0	Reserved
7-0	RXFILTERTHRESH	0-FFh	Receive filter low threshold. These bits contain the free buffer count threshold value for filtering low priority incoming frames. This field should remain 0, if no filtering is desired.

5.26 Receive Channel 0-7 Flow Control Threshold Register (RXnFLOWTHRESH)

The receive channel 0-7 flow control threshold register (RX*n*FLOWTHRESH) is shown in Figure 52 and described in Table 51.

Figure 52. Receive Channel *n* Flow Control Threshold Register (RX*n*FLOWTHRESH)



LEGEND: R = Read only; R/W = Read/Write; -*n* = value after reset

Table 51. Receive Channel *n* Flow Control Threshold Register (RX*n*FLOWTHRESH) Field Descriptions

Bit	Field	Value	Description
31-8	Reserved	0	Reserved
7-0	RX*n*FLOWTHRESH	0-FFh	Receive flow threshold. These bits contain the threshold value for issuing flow control on incoming frames for channel *n* (when enabled).

5.27 Receive Channel 0-7 Free Buffer Count Register (RXnFREEBUFFER)

The receive channel 0-7 free buffer count register (RX*n*FREEBUFFER) is shown in Figure 53 and described in Table 52.

Figure 53. Receive Channel *n* Free Buffer Count Register (RX*n*FREEBUFFER)

31	16
Reserved	
R-0	

15	0
RX*n*FREEBUF	
WI-0	

LEGEND: R = Read only; WI = Write to increment; -*n* = value after reset

Table 52. Receive Channel *n* Free Buffer Count Register (RX*n*FREEBUFFER) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved	0	Reserved
15-0	RX*n*FREEBUF	0-FFh	Receive free buffer count. These bits contain the count of free buffers available. The RXFILTERTHRESH value is compared with this field to determine if low priority frames should be filtered. The RX*n*FLOWTHRESH value is compared with this field to determine if receive flow control should be issued against incoming packets (if enabled). This is a write-to-increment field. This field rolls over to 0 on overflow. If hardware flow control or QOS is used, the host must initialize this field to the number of available buffers (one register per channel). The EMAC decrements the associated channel register for each received frame by the number of buffers in the received frame. The host must write this field with the number of buffers that have been freed due to host processing.

5.28 MAC Control Register (MACCONTROL)

The MAC control register (MACCONTROL) is shown in Figure 54 and described in Table 53.

Figure 54. MAC Control Register (MACCONTROL)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
R-0															

15	14	13	12	11	10	9	8
Reserved	RXOFFLENBLOCK	RXOWNERSHIP	Reserved	CMDIDLE	Reserved	TXPTYPE	Reserved
R-0	R/W-0	R/W-0	R-0	R/W-0	R-0	R/W-0	R-0

7	6	5	4	3	2	1	0
Reserved	TXPACE	GMIIEN	TXFLOWEN	RXBUFFERFLOWEN	Reserved	LOOPBACK	FULLDUPLEX
R-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R/W-0	R/W-0

LEGEND: R = Read only; R/W = Read/Write; *-n* = value after reset

Table 53. MAC Control Register (MACCONTROL) Field Descriptions

Bit	Field	Value	Description
31-15	Reserved	0	Reserved
14	RXOFFLENBLOCK		Receive offset / length word write block
		0	Do not block the DMA writes to the receive buffer descriptor offset / buffer length word.
		1	Block all EMAC DMA controller writes to the receive buffer descriptor offset / buffer length words during packet processing. When this bit is set, the EMAC will never write the third word to any receive buffer descriptor.
13	RXOWNERSHIP		Receive ownership write bit value
		0	The EMAC writes the Receive ownership bit to 0 at the end of packet processing.
		1	The EMAC writes the Receive ownership bit to 1 at the end of packet processing. If you do not use the ownership mechanism, you can set this mode to preclude the necessity of software having to set this bit each time the buffer descriptor is used.
12	Reserved	0	Reserved
11	CMDIDLE		Command Idle bit
		0	Idle not commanded
		1	Idle commanded (read IDLE in the MACSTATUS register)
10	Reserved	0	Reserved
9	TXPTYPE		Transmit queue priority type
		0	The queue uses a round-robin scheme to select the next channel for transmission.
		1	The queue uses a fixed-priority (channel 7 highest priority) scheme to select the next channel for transmission.
8-7	Reserved	0	Reserved
6	TXPACE		Transmit pacing enable bit
		0	Transmit pacing is disabled.
		1	Transmit pacing is enabled.
5	GMIIEN		GMII enable bit
		0	GMII RX and TX are held in reset.
		1	GMII RX and TX are enabled for receive and transmit.
4	TXFLOWEN		Transmit flow control enable bit. This bit determines if incoming pause frames are acted upon in full-duplex mode. Incoming pause frames are not acted upon in half-duplex mode, regardless of this bit setting. The RXMBPENABLE bits determine whether or not received pause frames are transferred to memory.
		0	Transmit flow control is disabled. Full-duplex mode: incoming pause frames are not acted upon.
		1	Transmit flow control is enabled. Full-duplex mode: incoming pause frames are acted upon.

Table 53. MAC Control Register (MACCONTROL) Field Descriptions (continued)

Bit	Field	Value	Description
3	RXBUFFERFLOWEN		Receive buffer flow control enable bit
		0	Receive flow control is disabled. Half-duplex mode: no flow control generated collisions are sent. Full-duplex mode: no outgoing pause frames are sent.
		1	Receive flow control is enabled. Half-duplex mode: collisions are initiated when receive buffer flow control is triggered. Full-duplex mode: outgoing pause frames are sent when receive flow control is triggered.
2	Reserved	0	Reserved
1	LOOPBACK		Loopback mode. The loopback mode forces internal full-duplex mode regardless of the FULLDUPLEX bit. The loopback bit should be changed only when GMIIEN bit is deasserted.
		0	Loopback mode is disabled.
		1	Loopback mode is enabled.
0	FULLDUPLEX		Full duplex mode.
		0	Half-duplex mode is enabled.
		1	Full-duplex mode is enabled.

5.29 MAC Status Register (MACSTATUS)

The MAC status register (MACSTATUS) is shown in Figure 55 and described in Table 54.

Figure 55. MAC Status Register (MACSTATUS)

31	30–24	23–20	19	18–16
IDLE	Reserved	TXERRCODE	Rsvd	TXERRCH
R-0	R-0	R-0	R-0	R-0

15–12	11	10–8
RXERRCODE	Reserved	RXERRCH
R-0	R-0	R-0

7–3	2	1	0
Reserved	RXQOSACT	RXFLOWACT	TXFLOWACT
R-0	R-0	R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

Table 54. MAC Status Register (MACSTATUS) Field Descriptions

Bit	Field	Value	Description
31	IDLE		EMAC idle bit. This bit is cleared to 0 at reset; one clock after reset, it goes to 1.
		0	The EMAC is not idle.
		1	The EMAC is in the idle state.
30-24	Reserved	0	Reserved
23-20	TXERRCODE	0-Fh	Transmit host error code. These bits indicate that EMAC detected transmit DMA related host errors. The host should read this field after a host error interrupt (HOSTPEND) to determine the error. Host error interrupts require hardware reset in order to recover. A 0 packet length is an error, but it is not detected.
		0	No error
		1h	SOP error; the buffer is the first buffer in a packet, but the SOP bit is not set in software.
		2h	Ownership bit not set in SOP buffer
		3h	Zero next buffer descriptor pointer without EOP
		4h	Zero buffer pointer
		5h	Zero buffer length
		6h	Packet length error (sum of buffers is less than packet length)
19	Reserved	0	Reserved
18-16	TXERRCH	0-7h	Transmit host error channel. These bits indicate which transmit channel the host error occurred on. This field is cleared to 0 on a host read.
		0	The host error occurred on transmit channel 0
		1h	The host error occurred on transmit channel 1
		2h	The host error occurred on transmit channel 2
		3h	The host error occurred on transmit channel 3
		4h	The host error occurred on transmit channel 4
		5h	The host error occurred on transmit channel 5
		6h	The host error occurred on transmit channel 6
		7h	The host error occurred on transmit channel 7
15-12	RXERRCODE	0-Fh	Receive host error code. These bits indicate that EMAC detected receive DMA related host errors. The host should read this field after a host error interrupt (HOSTPEND) to determine the error. Host error interrupts require hardware reset in order to recover.
		0	No error
		2h	Ownership bit not set in SOP buffer
		4h	Zero buffer pointer
11	Reserved	0	Reserved

Table 54. MAC Status Register (MACSTATUS) Field Descriptions (continued)

Bit	Field	Value	Description
10-8	RXERRCH	0-3h	Receive host error channel. These bits indicate which receive channel the host error occurred on. This field is cleared to 0 on a host read.
		0	The host error occurred on receive channel 0
		1h	The host error occurred on receive channel 1
		2h	The host error occurred on receive channel 2
		3h	The host error occurred on receive channel 3
		4h	The host error occurred on receive channel 4
		5h	The host error occurred on receive channel 5
		6h	The host error occurred on receive channel 6
		7h	The host error occurred on receive channel 7
7-3	Reserved	0	Reserved
2	RXQOSACT		Receive Quality of Service (QOS) active bit. When asserted, indicates that receive quality of service is enabled and that at least one channel freebuffer count (RX*n*FREEBUFFER) is less than or equal to the RXFILTERLOWTHRESH value.
		0	Receive quality of service is disabled.
		1	Receive quality of service is enabled.
1	RXFLOWACT		Receive flow control active bit. When asserted, at least one channel freebuffer count (RX*n*FREEBUFFER) is less than or equal to the channel's corresponding RX*n*FILTERTHRESH value.
		0	Receive flow control is inactive.
		1	Receive flow control is active.
0	TXFLOWACT		Transmit flow control active bit. When asserted, this bit indicates that the pause time period is being observed for a received pause frame. No new transmissions will begin while this bit is asserted, except for the transmission of pause frames. Any transmission in progress when this bit is asserted will complete.
		0	Transmit flow control is inactive.
		1	Transmit flow control is active.

5.30 Emulation Control Register (EMCONTROL)

The emulation control register (EMCONTROL) is shown in Figure 56 and described in Table 55.

Figure 56. Emulation Control Register (EMCONTROL)

31	16
Reserved	
R-0	

15	2	1	0
Reserved		SOFT	FREE
R-0		R/W-0	R/W-0

LEGEND: R = Read only; R/W = Read/Write; *-n* = value after reset

Table 55. Emulation Control Register (EMCONTROL) Field Descriptions

Bit	Field	Value	Description
31-2	Reserved	0	Reserved
1	SOFT		Emulation soft bit. This bit is used in conjunction with FREE bit to determine the emulation suspend mode. This bit has no effect if FREE = 1.
		0	Soft mode is disabled. EMAC stops immediately during emulation halt.
		1	Soft mode is enabled. During emulation halt, EMAC stops after completion of current operation.
0	FREE		Emulation free bit. This bit is used in conjunction with SOFT bit to determine the emulation suspend mode.
		0	Free-running mode is disabled. During emulation halt, SOFT bit determines operation of EMAC.
		1	Free-running mode is enabled. During emulation halt, EMAC continues to operate.

5.31 FIFO Control Register (FIFOCONTROL)

The FIFO control register (FIFOCONTROL) is shown in Figure 57 and described in Table 56.

Figure 57. FIFO Control Register (FIFOCONTROL)

31	16
Reserved	
R-0	

15	2	1	0
Reserved		TXCELLTHRESH	
R-0		R/W-2h	

LEGEND: R = Read only; R/W = Read/Write; *-n* = value after reset

Table 56. FIFO Control Register (FIFOCONTROL) Field Descriptions

Bit	Field	Value	Description
31-2	Reserved	0	Reserved
1-0	TXCELLTHRESH	0-3h	Transmit FIFO cell threshold. Indicates the number of 64-byte packet cells required to be in the transmit FIFO before the packet transfer is initiated. Packets with fewer cells will be initiated when the complete packet is contained in the FIFO. The default value is 2, but 3 is also valid. 0 and 1 are not valid values.
		0-1h	Not a valid value.
		2h	Two 64-byte packet cells required to be in the transmit FIFO.
		3h	Three 64-byte packet cells required to be in the transmit FIFO.

5.32 MAC Configuration Register (MACCONFIG)

The MAC configuration register (MACCONFIG) is shown in Figure 58 and described in Table 57.

Figure 58. MAC Configuration Register (MACCONFIG)

31	24	23	16
TXCELLDEPTH		RXCELLDEPTH	
R-3h		R-3h	

15	8	7	0
ADDRESSTYPE		MACCFIG	
R-1		R-1	

LEGEND: R = Read only; -*n* = value after reset

Table 57. MAC Configuration Register (MACCONFIG) Field Descriptions

Bit	Field	Value	Description
31-24	TXCELLDEPTH	3h	Transmit cell depth. These bits indicate the number of cells in the transmit FIFO.
23-16	RXCELLDEPTH	3h	Receive cell depth. These bits indicate the number of cells in the receive FIFO.
15-8	ADDRESSTYPE	1h	Address type
7-0	MACCFIG	1h	MAC configuration value

5.33 Soft Reset Register (SOFTRESET)

The soft reset register (SOFTRESET) is shown in Figure 59 and described in Table 58.

Figure 59. Soft Reset Register (SOFTRESET)

31	16
Reserved	
R-0	

15	1	0
Reserved		SOFTRESET
R-0		R/W-0

LEGEND: R = Read only; R/W = Read/Write; -*n* = value after reset

Table 58. Soft Reset Register (SOFTRESET) Field Descriptions

Bit	Field	Value	Description
31-1	Reserved	0	Reserved
0	SOFTRESET		Software reset. Writing a 1 to this bit causes the EMAC logic to be reset. Software reset occurs when the receive and transmit DMA controllers are in an idle state to avoid locking up the Configuration bus. After writing a 1 to this bit, it may be polled to determine if the reset has occurred. If a 1 is read, the reset has not yet occurred. If a 0 is read, then a reset has occurred.
		0	A software reset has not occurred.
		1	A software reset has occurred.

5.34 MAC Source Address Low Bytes Register (MACSRCADDRLO)

The MAC source address low bytes register (MACSRCADDRLO) is shown in Figure 60 and described in Table 59.

Figure 60. MAC Source Address Low Bytes Register (MACSRCADDRLO)

31	16
Reserved	
R-0	

15	8	7	0
MACSRCADDR0		MACSRCADDR1	
R/W-0		R/W-0	

LEGEND: R = Read only; R/W = Read/Write; -*n* = value after reset

Table 59. MAC Source Address Low Bytes Register (MACSRCADDRLO) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved	0	Reserved
15-8	MACSRCADDR0	0-FFh	MAC source address lower 8 bits (byte 0)
7-0	MACSRCADDR1	0-FFh	MAC source address bits 15-8 (byte 1)

5.35 MAC Source Address High Bytes Register (MACSRCADDRHI)

The MAC source address high bytes register (MACSRCADDRHI) is shown in Figure 61 and described in Table 60.

Figure 61. MAC Source Address High Bytes Register (MACSRCADDRHI)

31	24	23	16
MACSRCADDR2		MACSRCADDR3	
R/W-0		R/W-0	

15	8	7	0
MACSRCADDR4		MACSRCADDR5	
R/W-0		R/W-0	

LEGEND: R = Read only; R/W = Read/Write; -*n* = value after reset

Table 60. MAC Source Address High Bytes Register (MACSRCADDRHI) Field Descriptions

Bit	Field	Value	Description
31-24	MACSRCADDR2	0-FFh	MAC source address bits 23-16 (byte 2)
23-16	MACSRCADDR3	0-FFh	MAC source address bits 31-24 (byte 3)
15-8	MACSRCADDR4	0-FFh	MAC source address bits 39-32 (byte 4)
7-0	MACSRCADDR5	0-FFh	MAC source address bits 47-40 (byte 5)

5.36 MAC Hash Address Register 1 (MACHASH1)

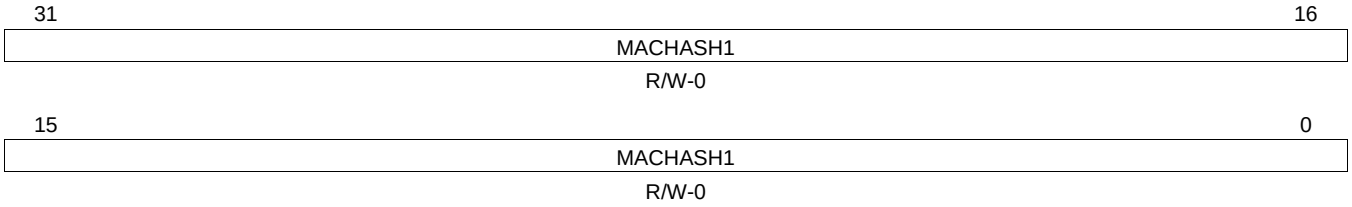
The MAC hash registers allow group addressed frames to be accepted on the basis of a hash function of the address. The hash function creates a 6-bit data value (Hash_fun) from the 48-bit destination address (DA) as follows:

```
Hash_fun(0)=DA(0) XOR DA(6) XOR DA(12) XOR DA(18) XOR DA(24) XOR DA(30) XOR DA(36) XOR DA(42);
Hash_fun(1)=DA(1) XOR DA(7) XOR DA(13) XOR DA(19) XOR DA(25) XOR DA(31) XOR DA(37) XOR DA(43);
Hash_fun(2)=DA(2) XOR DA(8) XOR DA(14) XOR DA(20) XOR DA(26) XOR DA(32) XOR DA(38) XOR DA(44);
Hash_fun(3)=DA(3) XOR DA(9) XOR DA(15) XOR DA(21) XOR DA(27) XOR DA(33) XOR DA(39) XOR DA(45);
Hash_fun(4)=DA(4) XOR DA(10) XOR DA(16) XOR DA(22) XOR DA(28) XOR DA(34) XOR DA(40) XOR DA(46);
Hash_fun(5)=DA(5) XOR DA(11) XOR DA(17) XOR DA(23) XOR DA(29) XOR DA(35) XOR DA(41) XOR DA(47);
```

This function is used as an offset into a 64-bit hash table stored in MACHASH1 and MACHASH2 that indicates whether a particular address should be accepted or not.

The MAC hash address register 1 (MACHASH1) is shown in Figure 62 and described in Table 61.

Figure 62. MAC Hash Address Register 1 (MACHASH1)



LEGEND: R = Read only; R/W = Read/Write; -*n* = value after reset

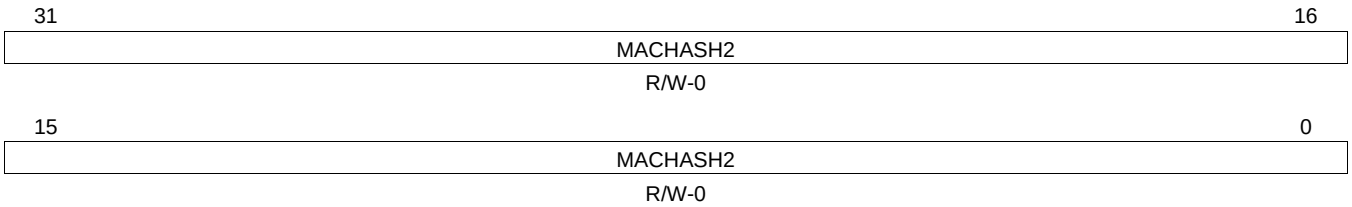
Table 61. MAC Hash Address Register 1 (MACHASH1) Field Descriptions

Bit	Field	Value	Description
31-0	MACHASH1	0-FFFF FFFFh	Least-significant 32 bits of the hash table corresponding to hash values 0 to 31. If a hash table bit is set, then a group address that hashes to that bit index is accepted.

5.37 MAC Hash Address Register 2 (MACHASH2)

The MAC hash address register 2 (MACHASH2) is shown in Figure 63 and described in Table 62.

Figure 63. MAC Hash Address Register 2 (MACHASH2)



LEGEND: R = Read only; R/W = Read/Write; -*n* = value after reset

Table 62. MAC Hash Address Register 2 (MACHASH2) Field Descriptions

Bit	Field	Value	Description
31-0	MACHASH2	0-FFFF FFFFh	Most-significant 32 bits of the hash table corresponding to hash values 32 to 63. If a hash table bit is set, then a group address that hashes to that bit index is accepted.

5.38 Back Off Test Register (BOFFTEST)

The back off test register (BOFFTEST) is shown in Figure 64 and described in Table 63.

Figure 64. Back Off Random Number Generator Test Register (BOFFTEST)

31	26	25	16
Reserved		RNDNUM	
R-0		R-0	

15	12	11	10	9	0
COLLCOUNT		Reserved		TXBACKOFF	
R-0		R-0		R-0	

LEGEND: R = Read only; -*n* = value after reset

Table 63. Back Off Test Register (BOFFTEST) Field Descriptions

Bit	Field	Value	Description
31-26	Reserved	0	Reserved
25-16	RNDNUM	0-3FFh	Backoff random number generator. This field allows the Backoff Random Number Generator to be read. Reading this field returns the generator's current value. The value is reset to 0 and begins counting on the clock after the deassertion of reset.
15-12	COLLCOUNT	0-Fh	Collision count. These bits indicate the number of collisions the current frame has experienced.
11-10	Reserved	0	Reserved
9-0	TXBACKOFF	0-3FFh	Backoff count. This field allows the current value of the backoff counter to be observed for test purposes. This field is loaded automatically according to the backoff algorithm, and is decremented by one for each slot time after the collision.

5.39 Transmit Pacing Algorithm Test Register (TPACETEST)

The transmit pacing algorithm test register (TPACETEST) is shown in Figure 65 and described in Table 64.

Figure 65. Transmit Pacing Algorithm Test Register (TPACETEST)

31	16
Reserved	
R-0	

15	5	4	0
Reserved		PACEVAL	
R-0		R-0	

LEGEND: R = Read only; -*n* = value after reset

Table 64. Transmit Pacing Algorithm Test Register (TPACETEST) Field Descriptions

Bit	Field	Value	Description
31-5	Reserved	0	Reserved
4-0	PACEVAL	0-1Fh	Pacing register current value. A nonzero value in this field indicates that transmit pacing is active. A transmit frame collision or deferral causes PACEVAL to be loaded with 1Fh (31); good frame transmissions (with no collisions or deferrals) cause PACEVAL to be decremented down to 0. When PACEVAL is nonzero, the transmitter delays four Inter Packet Gaps between new frame transmissions after each successfully transmitted frame that had no deferrals or collisions. If a transmit frame is deferred or suffers a collision, the IPG time is not stretched to four times the normal value. Transmit pacing helps reduce capture effects, which improves overall network bandwidth.

5.40 Receive Pause Timer Register (RXPAUSE)

The receive pause timer register (RXPAUSE) is shown in Figure 66 and described in Table 65.

Figure 66. Receive Pause Timer Register (RXPAUSE)

31	16
Reserved	
R-0	

15	0
PAUSETIMER	
R-0	

LEGEND: R = Read only; -*n* = value after reset

Table 65. Receive Pause Timer Register (RXPAUSE) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved	0	Reserved
15-0	PAUSETIMER	0-FFh	Receive pause timer value. These bits allow the contents of the receive pause timer to be observed. The receive pause timer is loaded with FF00h when the EMAC sends an outgoing pause frame (with pause time of FFFFh). The receive pause timer is decremented at slot time intervals. If the receive pause timer decrements to 0, then another outgoing pause frame is sent and the load/decrement process is repeated.

5.41 Transmit Pause Timer Register (TXPAUSE)

The transmit pause timer register (TXPAUSE) is shown in Figure 67 and described in Table 66.

Figure 67. Transmit Pause Timer Register (TXPAUSE)

31	16
Reserved	
R-0	

15	0
PAUSETIMER	
R-0	

LEGEND: R = Read only; -*n* = value after reset

Table 66. Transmit Pause Timer Register (TXPAUSE) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved	0	Reserved
15-0	PAUSETIMER	0-FFh	Transmit pause timer value. These bits allow the contents of the transmit pause timer to be observed. The transmit pause timer is loaded by a received (incoming) pause frame, and then decremented at slot time intervals down to 0, at which time EMAC transmit frames are again enabled.

5.42 MAC Address Low Bytes Register (MACADDRLO)

The MAC address low bytes register used in address matching (MACADDRLO), is shown in Figure 68 and described in Table 67.

Figure 68. MAC Address Low Bytes Register (MACADDRLO)

31	16
Reserved	
R-0	

15	8	7	0
MACADDR0		MACADDR1	
R/W-0		R/W-0	

LEGEND: R = Read only; R/W = Read/Write; -*n* = value after reset

Table 67. MAC Address Low Bytes Register (MACADDRLO) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved	0	Reserved
15-8	MACADDR0	0-FFh	MAC address lower 8 bits (byte 0)
7-0	MACADDR1	0-FFh	MAC address bits 15-8 (byte 1)

5.43 MAC Address High Bytes Register (MACADDRHI)

The MAC address high bytes register (MACADDRHI) is shown in Figure 69 and described in Table 68.

Figure 69. MAC Address High Bytes Register (MACADDRHI)

31	24	23	16
MACADDR2		MACADDR3	
R/W-0		R/W-0	

15	8	7	0
MACADDR4		MACADDR5	
R/W-0		R/W-0	

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 68. MAC Address High Bytes Register (MACADDRHI) Field Descriptions

Bit	Field	Value	Description
31-24	MACADDR2	0-FFh	MAC source address bits 23-16 (byte 2)
23-16	MACADDR3	0-FFh	MAC source address bits 31-24 (byte 3)
15-8	MACADDR4	0-FFh	MAC source address bits 39-32 (byte 4)
7-0	MACADDR5	0-FFh	MAC source address bits 47-40 (byte 5). Bit 40 is the group bit. It is forced to 0 and read as 0. Therefore, only unicast addresses are represented in the address table.

5.44 MAC Index Register (MACINDEX)

The MAC index register (MACINDEX) is shown in Figure 70 and described in Table 69.

Figure 70. MAC Index Register (MACINDEX)

31	16
Reserved	
R-0	

15	3	2	0
Reserved		MACINDEX	
R-0		R/W-0	

LEGEND: R = Read only; R/W = Read/Write; *-n* = value after reset

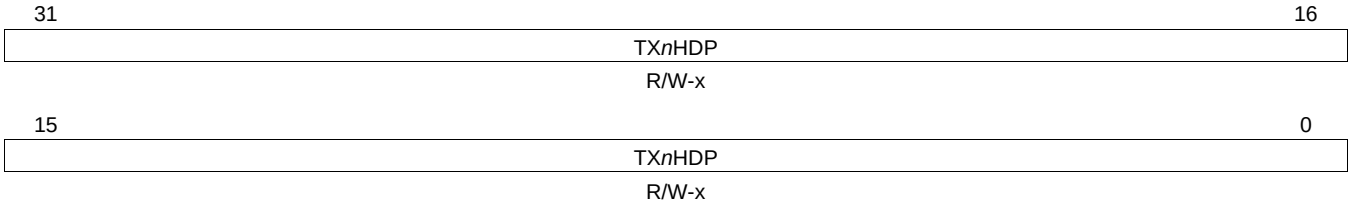
Table 69. MAC Index Register (MACINDEX) Field Descriptions

Bit	Field	Value	Description
31-3	Reserved	0	Reserved
2-0	MACINDEX	0-7h	MAC address index. All eight addresses share the upper 40 bits. Only the lower byte is unique for each address. An address is written by first writing the address number (channel) into the MACINDEX register. The upper 32 bits of the address are then written to the MACADDRHI register, which is followed by writing the lower 16 bits of the address to the MACADDRLO register. Since all eight addresses share the upper 40 bits of the address, the MACADDRHI register only needs to be written the first time.

5.45 Transmit Channel 0-7 DMA Head Descriptor Pointer Register (TXnHDP)

The transmit channel 0-7 DMA head descriptor pointer register (TX*n*HDP) is shown in Figure 71 and described in Table 70.

Figure 71. Transmit Channel *n* DMA Head Descriptor Pointer Register (TX*n*HDP)



LEGEND: R/W = Read/Write; -*n* = value after reset; -x = value is indeterminate after reset

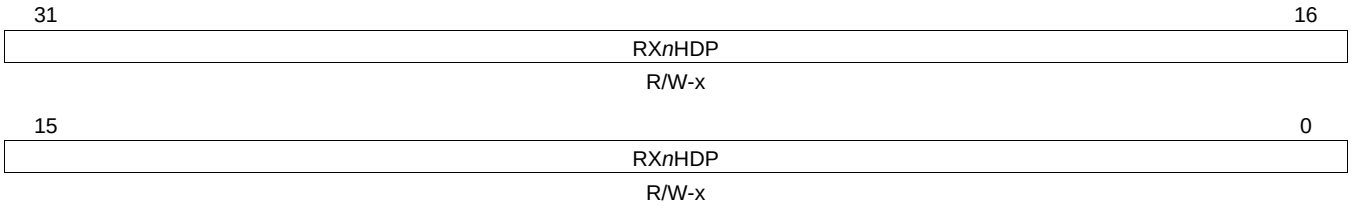
Table 70. Transmit Channel *n* DMA Head Descriptor Pointer Register (TX*n*HDP) Field Descriptions

Bit	Field	Value	Description
31-0	TX*n*HDP	0-FFFF FFFFh	Transmit channel *n* DMA Head Descriptor pointer. Writing a transmit DMA buffer descriptor address to a head pointer location initiates transmit DMA operations in the queue for the selected channel. Writing to these locations when they are nonzero is an error (except at reset). Host software must initialize these locations to 0 on reset.

5.46 Receive Channel 0-7 DMA Head Descriptor Pointer Register (RXnHDP)

The receive channel 0-7 DMA head descriptor pointer register (RX*n*HDP) is shown in Figure 72 and described in Table 71.

Figure 72. Receive Channel *n* DMA Head Descriptor Pointer Register (RX*n*HDP)



LEGEND: R/W = Read/Write; -*n* = value after reset; -x = value is indeterminate after reset

Table 71. Receive Channel *n* DMA Head Descriptor Pointer Register (RX*n*HDP) Field Descriptions

Bit	Field	Value	Description
31-0	RX*n*HDP	0-FFFF FFFFh	Receive channel *n* DMA Head Descriptor pointer. Writing a receive DMA buffer descriptor address to this location allows receive DMA operations in the selected channel when a channel frame is received. Writing to these locations when they are nonzero is an error (except at reset). Host software must initialize these locations to 0 on reset.

5.47 Transmit Channel 0-7 Completion Pointer Register (TXnCP)

The transmit channel 0-7 completion pointer register (TX*n*CP) is shown in Figure 73 and described in Table 72.

Figure 73. Transmit Channel *n* Completion Pointer Register (TX*n*CP)

31	16
TX*n*CP	
R/W-x	

15	0
TX*n*CP	
R/W-x	

LEGEND: R/W = Read/Write; -*n* = value after reset; -x = value is indeterminate after reset

Table 72. Transmit Channel *n* Completion Pointer Register (TX*n*CP) Field Descriptions

Bit	Field	Value	Description
31-0	TX*n*CP	0-FFFF FFFFh	Transmit channel *n* completion pointer register is written by the host with the buffer descriptor address for the last buffer processed by the host during interrupt processing. The EMAC uses the value written to determine if the interrupt should be deasserted.

5.48 Receive Channel 0-7 Completion Pointer Register (RXnCP)

The receive channel 0-7 completion pointer register (RX*n*CP) is shown in Figure 74 and described in Table 73.

Figure 74. Receive Channel *n* Completion Pointer Register (RX*n*CP)

31	16
RX*n*CP	
R/W-x	

15	0
RX*n*CP	
R/W-x	

LEGEND: R/W = Read/Write; -*n* = value after reset; -x = value is indeterminate after reset

Table 73. Receive Channel *n* Completion Pointer Register (RX*n*CP) Field Descriptions

Bit	Field	Value	Description
31-0	RX*n*CP	0-FFFF FFFFh	Receive channel *n* completion pointer register is written by the host with the buffer descriptor address for the last buffer processed by the host during interrupt processing. The EMAC uses the value written to determine if the interrupt should be deasserted.

5.49 Network Statistics Registers

The EMAC has a set of statistics that record events associated with frame traffic. The statistics values are cleared to zero 38 clocks after the rising edge of reset. When the GMIIEN bit in the MACCONTROL register is set, all statistics registers (see Figure 75) are write-to-decrement. The value written is subtracted from the register value with the result stored in the register. If a value greater than the statistics value is written, then zero is written to the register (writing FFFF FFFFh clears a statistics location). When the GMIIEN bit is cleared, all statistics registers are read/write (normal write direct, so writing 0000 0000h clears a statistics location). All write accesses must be 32-bit accesses.

The statistics interrupt (STATPEND) is issued, if enabled, when any statistics value is greater than or equal to 8000 0000h. The statistics interrupt is removed by writing to decrement any statistics value greater than 8000 0000h. The statistics are mapped into internal memory space and are 32-bits wide. All statistics rollover from FFFF FFFFh to 0000 0000h.

Figure 75. Statistics Register

31	16
COUNT	
R/WD-0	

15	0
COUNT	
R/WD-0	

LEGEND: R/W = Read/Write; WD = Write to decrement; -*n* = value after reset

5.49.1 Good Receive Frames Register (RXGOODFRAMES)

The total number of good frames received on the EMAC. A good frame is defined as having all of the following:

- Any data or MAC control frame that matched a unicast, broadcast, or multicast address, or matched due to promiscuous mode
- Was of length 64 to RXMAXLEN bytes inclusive
- Had no CRC error, alignment error, or code error

See **Section 2.5.5** for definitions of alignment, code, and CRC errors. Overruns have no effect on this statistic.

5.49.2 Broadcast Receive Frames Register (RXBCASTFRAMES)

The total number of good broadcast frames received on the EMAC. A good broadcast frame is defined as having all of the following:

- Any data or MAC control frame that was destined for address FF-FF-FF-FF-FF-FFh only
- Was of length 64 to RXMAXLEN bytes inclusive
- Had no CRC error, alignment error, or code error

See Section 2.5.5 for definitions of alignment, code, and CRC errors. Overruns have no effect on this statistic.

5.49.3 Multicast Receive Frames Register (RXMCASTFRAMES)

The total number of good multicast frames received on the EMAC. A good multicast frame is defined as having all of the following:

- Any data or MAC control frame that was destined for any multicast address other than FF-FF-FF-FF-FF-FFh
- Was of length 64 to RXMAXLEN bytes inclusive
- Had no CRC error, alignment error, or code error

See Section 2.5.5 for definitions of alignment, code, and CRC errors. Overruns have no effect on this statistic.

5.49.4 Pause Receive Frames Register (RXPAUSEFRAMES)

The total number of IEEE 802.3X pause frames received by the EMAC (whether acted upon or not). A pause frame is defined as having all of the following:

- Contained any unicast, broadcast, or multicast address
- Contained the length/type field value 88.08h and the opcode 0001h
- Was of length 64 to RXMAXLEN bytes inclusive
- Had no CRC error, alignment error, or code error
- Pause-frames had been enabled on the EMAC (TXFLOWEN bit is set in MACCONTROL).

The EMAC could have been in either half-duplex or full-duplex mode. See Section 2.5.5 for definitions of alignment, code, and CRC errors. Overruns have no effect on this statistic.

5.49.5 Receive CRC Errors Register (RXCRCERRORS)

The total number of frames received on the EMAC that experienced a CRC error. A frame with CRC errors is defined as having all of the following:

- Was any data or MAC control frame that matched a unicast, broadcast, or multicast address, or matched due to promiscuous mode
- Was of length 64 to RXMAXLEN bytes inclusive
- Had no alignment or code error
- Had a CRC error. A CRC error is defined as having all of the following:
  - A frame containing an even number of nibbles
  - Fails the frame check sequence test

See Section 2.5.5 for definitions of alignment, code, and CRC errors. Overruns have no effect on this statistic.

5.49.6 Receive Alignment/Code Errors Register (RXALIGNCODEERRORS)

The total number of frames received on the EMAC that experienced an alignment error or code error. Such a frame is defined as having all of the following:

- Was any data or MAC control frame that matched a unicast, broadcast, or multicast address, or matched due to promiscuous mode
- Was of length 64 to RXMAXLEN bytes inclusive
- Had either an alignment error or a code error
  - An alignment error is defined as having all of the following:
    - A frame containing an odd number of nibbles
    - Fails the frame check sequence test, if the final nibble is ignored
  - A code error is defined as a frame that has been discarded because the EMACs MRXER pin is driven with a one for at least one bit-time's duration at any point during the frame's reception.

Overruns have no effect on this statistic.

CRC alignment or code errors can be calculated by summing receive alignment errors, receive code errors, and receive CRC errors.

5.49.7 Receive Oversized Frames Register (RXOVERSIZED)

The total number of oversized frames received on the EMAC. An oversized frame is defined as having all of the following:

- Was any data or MAC control frame that matched a unicast, broadcast, or multicast address, or matched due to promiscuous mode
- Was greater than RXMAXLEN in bytes
- Had no CRC error, alignment error, or code error

See Section 2.5.5 for definitions of alignment, code, and CRC errors. Overruns have no effect on this statistic.

5.49.8 Receive Jabber Frames Register (RXJABBER)

The total number of jabber frames received on the EMAC. A jabber frame is defined as having all of the following:

- Was any data or MAC control frame that matched a unicast, broadcast, or multicast address, or matched due to promiscuous mode
- Was greater than RXMAXLEN bytes long
- Had a CRC error, alignment error, or code error

See Section 2.5.5 for definitions of alignment, code, and CRC errors. Overruns have no effect on this statistic.

5.49.9 Receive Undersized Frames Register (RXUNDERSIZED)

The total number of undersized frames received on the EMAC. An undersized frame is defined as having all of the following:

- Was any data frame that matched a unicast, broadcast, or multicast address, or matched due to promiscuous mode
- Was less than 64 bytes long
- Had no CRC error, alignment error, or code error

See Section 2.5.5 for definitions of alignment, code, and CRC errors. Overruns have no effect on this statistic.

5.49.10 Receive Frame Fragments Register (RXFRAGMENTS)

The total number of frame fragments received on the EMAC. A frame fragment is defined as having all of the following:

- Any data frame (address matching does not matter)
- Was less than 64 bytes long
- Had a CRC error, alignment error, or code error
- Was not the result of a collision caused by half duplex, collision based flow control

See Section 2.5.5 for definitions of alignment, code, and CRC errors. Overruns have no effect on this statistic.

5.49.11 Filtered Receive Frames Register (RXFILTERED)

The total number of frames received on the EMAC that the EMAC address matching process indicated should be discarded. Such a frame is defined as having all of the following:

- Was any data frame (not MAC control frame) destined for any unicast, broadcast, or multicast address
- Did not experience any CRC error, alignment error, code error
- The address matching process decided that the frame should be discarded (filtered) because it did not match the unicast, broadcast, or multicast address, and it did not match due to promiscuous mode.

To determine the number of receive frames discarded by the EMAC for any reason, sum the following statistics (promiscuous mode disabled):

- Receive fragments
- Receive undersized frames
- Receive CRC errors
- Receive alignment/code errors
- Receive jabbers
- Receive overruns
- Receive filtered frames

This may not be an exact count because the receive overruns statistic is independent of the other statistics, so if an overrun occurs at the same time as one of the other discard reasons, then the above sum double-counts that frame.

5.49.12 Receive QOS Filtered Frames Register (RXQOSFILTERED)

The total number of frames received on the EMAC that were filtered due to receive quality of service (QOS) filtering. Such a frame is defined as having all of the following:

- Any data or MAC control frame that matched a unicast, broadcast, or multicast address, or matched due to promiscuous mode
- The frame destination channel flow control threshold register (RX*n*FLOWTHRESH) value was greater than or equal to the channel's corresponding free buffer register (RX*n*FREEBUFFER) value
- Was of length 64 to RXMAXLEN
- RXQOSEN bit is set in RXMBPENABLE
- Had no CRC error, alignment error, or code error

See Section 2.5.5 for definitions of alignment, code, and CRC errors. Overruns have no effect on this statistic.

5.49.13 Receive Octet Frames Register (RXOCTETS)

The total number of bytes in all good frames received on the EMAC. A good frame is defined as having all of the following:

- Any data or MAC control frame that matched a unicast, broadcast, or multicast address, or matched due to promiscuous mode
- Was of length 64 to RXMAXLEN bytes inclusive
- Had no CRC error, alignment error, or code error

See Section 2.5.5 for definitions of alignment, code, and CRC errors. Overruns have no effect on this statistic.

5.49.14 Good Transmit Frames Register (TXGOODFRAMES)

The total number of good frames transmitted on the EMAC. A good frame is defined as having all of the following:

- Any data or MAC control frame that was destined for any unicast, broadcast, or multicast address
- Was any length
- Had no late or excessive collisions, no carrier loss, and no underrun

5.49.15 Broadcast Transmit Frames Register (TXBCASTFRAMES)

The total number of good broadcast frames transmitted on the EMAC. A good broadcast frame is defined as having all of the following:

- Any data or MAC control frame destined for address FF-FF-FF-FF-FF-FFh only
- Was of any length
- Had no late or excessive collisions, no carrier loss, and no underrun

5.49.16 Multicast Transmit Frames Register (TXMCASTFRAMES)

The total number of good multicast frames transmitted on the EMAC. A good multicast frame is defined as having all of the following:

- Any data or MAC control frame destined for any multicast address other than FF-FF-FF-FF-FF-FFh
- Was of any length
- Had no late or excessive collisions, no carrier loss, and no underrun

5.49.17 Pause Transmit Frames Register (TXPAUSEFRAMES)

The total number of IEEE 802.3X pause frames transmitted by the EMAC. Pause frames cannot underrun or contain a CRC error because they are created in the transmitting MAC, so these error conditions have no effect on this statistic. Pause frames sent by software are not included in this count. Since pause frames are only transmitted in full-duplex mode, carrier loss and collisions have no effect on this statistic.

Transmitted pause frames are always 64-byte multicast frames so appear in the multicast transmit frames register and 64 octect frames register statistics.

5.49.18 Deferred Transmit Frames Register (TXDEFERRED)

The total number of frames transmitted on the EMAC that first experienced deferment. Such a frame is defined as having all of the following:

- Was any data or MAC control frame destined for any unicast, broadcast, or multicast address
- Was any size
- Had no carrier loss and no underrun
- Experienced no collisions before being successfully transmitted
- Found the medium busy when transmission was first attempted, so had to wait.

CRC errors have no effect on this statistic.

5.49.19 Transmit Collision Frames Register (TXCOLLISION)

The total number of times that the EMAC experienced a collision. Collisions occur under two circumstances:

- When a transmit data or MAC control frame has all of the following:
  - Was destined for any unicast, broadcast, or multicast address
  - Was any size
  - Had no carrier loss and no underrun
  - Experienced a collision. A jam sequence is sent for every non-late collision, so this statistic increments on each occasion if a frame experiences multiple collisions (and increments on late collisions).
- When the EMAC is in half-duplex mode, flow control is active, and a frame reception begins.

CRC errors have no effect on this statistic.

5.49.20 Transmit Single Collision Frames Register (TXSINGLECOLL)

The total number of frames transmitted on the EMAC that experienced exactly one collision. Such a frame is defined as having all of the following:

- Was any data or MAC control frame destined for any unicast, broadcast, or multicast address
- Was any size
- Had no carrier loss and no underrun
- Experienced one collision before successful transmission. The collision was not late.

CRC errors have no effect on this statistic.

5.49.21 Transmit Multiple Collision Frames Register (TXMULTICOLL)

The total number of frames transmitted on the EMAC that experienced multiple collisions. Such a frame is defined as having all of the following:

- Was any data or MAC control frame destined for any unicast, broadcast, or multicast address
- Was any size
- Had no carrier loss and no underrun
- Experienced 2 to 15 collisions before being successfully transmitted. None of the collisions were late.

CRC errors have no effect on this statistic.

5.49.22 Transmit Excessive Collision Frames Register (TXEXCESSIVECOLL)

The total number of frames when transmission was abandoned due to excessive collisions. Such a frame is defined as having all of the following:

- Was any data or MAC control frame destined for any unicast, broadcast, or multicast address
- Was any size
- Had no carrier loss and no underrun
- Experienced 16 collisions before abandoning all attempts at transmitting the frame. None of the collisions were late.

CRC errors have no effect on this statistic.

5.49.23 Transmit Late Collision Frames Register (TXLATECOLL)

The total number of frames when transmission was abandoned due to a late collision. Such a frame is defined as having all of the following:

- Was any data or MAC control frame destined for any unicast, broadcast, or multicast address
- Was any size
- Had no carrier loss and no underrun
- Experienced a collision later than 512 bit-times into the transmission. There may have been up to 15 previous (non-late) collisions that had previously required the transmission to be reattempted. The late collisions statistic dominates over the single, multiple, and excessive collisions statistics. If a late collision occurs, the frame is not counted in any of these other three statistics.

CRC errors, carrier loss, and underrun have no effect on this statistic.

5.49.24 Transmit Underrun Error Register (TXUNDERRUN)

The number of frames sent by the EMAC that experienced FIFO underrun. Late collisions, CRC errors, carrier loss, and underrun have no effect on this statistic.

5.49.25 Transmit Carrier Sense Errors Register (TXCARRIERSENSE)

The total number of frames on the EMAC that experienced carrier loss. Such a frame is defined as having all of the following:

- Was any data or MAC control frame destined for any unicast, broadcast, or multicast address
- Was any size
- The carrier sense condition was lost or never asserted when transmitting the frame (the frame is not retransmitted)

CRC errors and underrun have no effect on this statistic.

5.49.26 Transmit Octet Frames Register (TXOCTETS)

The total number of bytes in all good frames transmitted on the EMAC. A good frame is defined as having all of the following:

- Any data or MAC control frame that was destined for any unicast, broadcast, or multicast address
- Was any length
- Had no late or excessive collisions, no carrier loss, and no underrun

5.49.27 Transmit and Receive 64 Octet Frames Register (FRAME64)

The total number of 64-byte frames received and transmitted on the EMAC. Such a frame is defined as having all of the following:

- Any data or MAC control frame that was destined for any unicast, broadcast, or multicast address
- Did not experience late collisions, excessive collisions, underrun, or carrier sense error
- Was exactly 64-bytes long. (If the frame was being transmitted and experienced carrier loss that resulted in a frame of this size being transmitted, then the frame is recorded in this statistic).

CRC errors, alignment/code errors, and overruns do not affect the recording of frames in this statistic.

5.49.28 Transmit and Receive 65 to 127 Octet Frames Register (FRAME65T127)

The total number of 65-byte to 127-byte frames received and transmitted on the EMAC. Such a frame is defined as having all of the following:

- Any data or MAC control frame that was destined for any unicast, broadcast, or multicast address
- Did not experience late collisions, excessive collisions, underrun, or carrier sense error
- Was 65-bytes to 127-bytes long

CRC errors, alignment/code errors, underruns, and overruns do not affect the recording of frames in this statistic.

5.49.29 Transmit and Receive 128 to 255 Octet Frames Register (FRAME128T255)

The total number of 128-byte to 255-byte frames received and transmitted on the EMAC. Such a frame is defined as having all of the following:

- Any data or MAC control frame that was destined for any unicast, broadcast, or multicast address
- Did not experience late collisions, excessive collisions, underrun, or carrier sense error
- Was 128-bytes to 255-bytes long

CRC errors, alignment/code errors, underruns, and overruns do not affect the recording of frames in this statistic.

5.49.30 Transmit and Receive 256 to 511 Octet Frames Register (FRAME256T511)

The total number of 256-byte to 511-byte frames received and transmitted on the EMAC. Such a frame is defined as having all of the following:

- Any data or MAC control frame that was destined for any unicast, broadcast, or multicast address
- Did not experience late collisions, excessive collisions, underrun, or carrier sense error
- Was 256-bytes to 511-bytes long

CRC errors, alignment/code errors, underruns, and overruns do not affect the recording of frames in this statistic.

5.49.31 Transmit and Receive 512 to 1023 Octet Frames Register (FRAME512T1023)

The total number of 512-byte to 1023-byte frames received and transmitted on the EMAC. Such a frame is defined as having all of the following:

- Any data or MAC control frame that was destined for any unicast, broadcast, or multicast address
- Did not experience late collisions, excessive collisions, underrun, or carrier sense error
- Was 512-bytes to 1023-bytes long

CRC errors, alignment/code errors, and overruns do not affect the recording of frames in this statistic.

5.49.32 Transmit and Receive 1024 to RXMAXLEN Octet Frames Register (FRAME1024TUP)

The total number of 1024-byte to RXMAXLEN-byte frames received and transmitted on the EMAC. Such a frame is defined as having all of the following:

- Any data or MAC control frame that was destined for any unicast, broadcast, or multicast address
- Did not experience late collisions, excessive collisions, underrun, or carrier sense error
- Was 1024-bytes to RXMAXLEN-bytes long

CRC/alignment/code errors, underruns, and overruns do not affect frame recording in this statistic.

5.49.33 Network Octet Frames Register (NETOCTETS)

The total number of bytes of frame data received and transmitted on the EMAC. Each frame counted has all of the following:

- Was any data or MAC control frame destined for any unicast, broadcast, or multicast address (address match does not matter)
- Was of any size (including less than 64-byte and greater than RXMAXLEN-byte frames)

Also counted in this statistic is:

- Every byte transmitted before a carrier-loss was experienced
- Every byte transmitted before each collision was experienced (multiple retries are counted each time)
- Every byte received if the EMAC is in half-duplex mode until a jam sequence was transmitted to initiate flow control. (The jam sequence is not counted to prevent double-counting).

Error conditions such as alignment errors, CRC errors, code errors, overruns, and underruns do not affect the recording of bytes in this statistic. The objective of this statistic is to give a reasonable indication of Ethernet utilization.

5.49.34 Receive FIFO or DMA Start of Frame Overruns Register (RXSOFOVERRUNS)

The total number of frames received on the EMAC that had either a FIFO or DMA start of frame (SOF) overrun. An SOF overrun frame is defined as having all of the following:

- Was any data or MAC control frame that matched a unicast, broadcast, or multicast address, or matched due to promiscuous mode
- Was of any size (including less than 64-byte and greater than RXMAXLEN-byte frames)
- The EMAC was unable to receive it because it did not have the resources to receive it (cell FIFO full or no DMA buffer available at the start of the frame).

CRC errors, alignment errors, and code errors have no effect on this statistic.

5.49.35 Receive FIFO or DMA Middle of Frame Overruns Register (RXMOFOVERRUNS)

The total number of frames received on the EMAC that had either a FIFO or DMA middle of frame (MOF) overrun. An MOF overrun frame is defined as having all of the following:

- Was any data or MAC control frame that matched a unicast, broadcast, or multicast address, or matched due to promiscuous mode
- Was of any size (including less than 64-byte and greater than RXMAXLEN-byte frames)
- The EMAC was unable to receive it because it did not have the resources to receive it (cell FIFO full or no DMA buffer available after the frame was successfully started - no SOF overrun).

CRC errors, alignment errors, and code errors have no effect on this statistic.

5.49.36 Receive DMA Overruns Register (RXDMAOVERRUNS)

The total number of frames received on the EMAC that had either a DMA start of frame (SOF) overrun or a DMA middle of frame (MOF) overrun. A receive DMA overrun frame is defined as having all of the following:

- Was any data or MAC control frame that matched a unicast, broadcast, or multicast address, or matched due to promiscuous mode
- Was of any size (including less than 64-byte and greater than RXMAXLEN-byte frames)
- The EMAC was unable to receive it because it did not have the DMA buffer resources to receive it (zero head descriptor pointer at the start or during the middle of the frame reception).

CRC errors, alignment errors, and code errors have no effect on this statistic.

Appendix A Glossary

Broadcast MAC Address — A special Ethernet MAC address used to send data to all Ethernet devices on the local network. The broadcast address is FFh-FFh-FFh-FFh-FFh-FFh. The LSB of the first byte is odd, qualifying it as a group address; however, its value is reserved for broadcast. It is classified separately by the EMAC.

Descriptor (Packet Buffer Descriptor) — A small memory structure that describes a larger block of memory in terms of size, location, and state. Descriptors are used by the EMAC and application to describe the memory buffers that hold Ethernet data.

Device — In this document, device refers to the TMS320DM643x processor.

Ethernet MAC Address (MAC Address) — A unique 6-byte address that identifies an Ethernet device on the network. In an Ethernet packet, a MAC address is used twice, first to identify the packet's destination, and second to identify the packet's sender or source. An Ethernet MAC address is normally specified in hexadecimal, using dashes to separate bytes. For example, 08h-00h-28h-32h-17h-42h.

The first three bytes normally designate the manufacturer of the device. However, when the first byte of the address is odd (LSB is 1), the address is a group address (broadcast or multicast). The second bit specifies whether the address is globally or locally administrated (not considered in this document).

Ethernet Packet (Packet) — An Ethernet packet is the collection of bytes that represents the data portion of a single Ethernet frame on the wire.

Full Duplex — Full-duplex operation allows simultaneous communication between a pair of stations using point-to-point media (dedicated channel). Full-duplex operation does not require that transmitters defer, nor do they monitor or react to receive activity, as there is no contention for a shared medium in this mode. Full-duplex mode can only be used when all of the following are true:

- The physical medium is capable of supporting simultaneous transmission and reception without interference.
- There are exactly two stations connected with a full duplex point-to-point link. As there is no contention for use of a shared medium, the multiple access (that is, CSMA/CD) algorithms are unnecessary.
- Both stations on the LAN are capable of, and have been configured to use, full-duplex operation.

The most common configuration envisioned for full-duplex operation consists of a central bridge (also known as a switch) with a dedicated LAN connecting each bridge port to a single device.

Full-duplex operation constitutes a proper subset of the MAC functionality required for half-duplex operation.

Half Duplex — In half-duplex mode, the CSMA/CD media access method is the means by which two or more stations share a common transmission medium. To transmit, a station waits (defers) for a quiet period on the medium, that is, no other station is transmitting. It then sends the intended message in bit-serial form. If, after initiating a transmission, the message collides with that of another station, then each transmitting station intentionally transmits for an additional predefined period to ensure propagation of the collision throughout the system. The station remains silent for a random amount of time (backoff) before attempting to transmit again.

Host — The host is an intelligent system resource that configures and manages each communications control module. The host is responsible for allocating memory, initializing all data structures, and responding to port (EMAC) interrupts. In this document, host refers to the TMS320DM643x device.

Jabber — A condition wherein a station transmits for a period of time longer than the maximum permissible packet length, usually due to a fault condition.

Link — The transmission path between any two instances of generic cabling.

Multicast MAC Address — A class of MAC address that sends a packet to potentially more than one recipient. A group address is specified by setting the LSB of the first MAC address byte to 1. Thus, 01h-02h-03h-04h-05h-06h is a valid multicast address. Typically, an Ethernet MAC looks for only certain multicast addresses on a network to reduce traffic load. The multicast address list of acceptable packets is specified by the application.

Physical Layer and Media Notation — To identify different Ethernet technologies, a simple, three-field, type notation is used. The Physical Layer type used by the Ethernet is specified by these fields:

<data rate in Mb/s><medium type><maximum segment length (×100m)>

The definitions for the technologies mentioned in this document are in Table A-1.

Table A-1. Physical Layer Definitions

Term	Definition
10Base-T	IEEE 802.3 Physical Layer specification for a 10 Mb/s CSMA/CD local area network over two pairs of twisted-pair telephone wire.
100Base-T	IEEE 802.3 Physical Layer specification for a 100 Mb/s CSMA/CD local area network over two pairs of Category 5 unshielded twisted-pair (UTP) or shielded twisted-pair (STP) wire.
Twisted pair	A cable element that consists of two insulated conductors twisted together in a regular fashion to form a balanced transmission line.

Port — Ethernet device.

Promiscuous Mode — EMAC receives frames that do not match its address.

Appendix B Revision History

Table B-1 lists the changes made since the previous version of this document.

Table B-1. Document Revision History

Reference	Additions/Modifications/Deletions
Section 2.4.1	Changed last sentence.
Table 2	Changed Data field Bytes and Description.
Table 25	Changed Register Description for FRAME1024TUP.
Section 5.49.32	Changed subsection.
Section 5.49.33	Changed second bulleted item.
Section 5.49.34	Changed second bulleted item.
Section 5.49.35	Changed second bulleted item.
Section 5.49.36	Changed second bulleted item.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
Low Power Wireless	www.ti.com/lpw	Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265